Towards Accurate and Scalable Recommender Systems

Manuel Pozo

CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS | le cnam

EDITE - École Doctorale du Conservatoire National des Arts et Métiers

Laboratoire CEDRIC - Centre d'Études et De Recherces en Informatiques et Communications

THÈSE DE DOCTORAT

présentée par : **Manuel POZO**

soutenue le : **12 octobre 2016**

pour obtenir le grade de : **Docteur du Conservatoire National des Arts et Métiers**

Discipline / Spécialité : **Informatique**

Towards Accurate and Scalable Recommender Systems

THÈSE DIRIGÉE PAR

Mme. METAIS Elisabeth — *Professeur , CNAM*

THÈSE ENCADRÉE PAR

Mme. CHIKY Raja — *Enseignant-chercheur, ISEP*

RAPPORTEURS

Mme. BOYER Anne — *Professeur, Université de Lorraine*
Mme. TEISSEIRE Maguelonne — *Directrice de recherche, IRSTEA*

EXAMINATEURS

M. MEZIANE Farid — *Professeur, Salford University*
M. DU MOUZA Cédric — *Maître de Conférences-HDR, CNAM*
M. MASSEGLIA Florent — *Chercheur-HDR, Université Montpellier 2*

PRÉSIDENT DU JURY

M. SENELLART Pierre — *Professeur, École Normale Supérieure*

Acknowledgments

For this dissertation, I would like to thank first of all Raja Chiky. She was the first person in my professional, she took an non-experienced Spanish student and give him trust and the possibility to learn and grow. I would not be here today without her support and confidence. I thank my thesis director, Elisabeth Métais, for her encourage during these years and the intern-ship opportunity she offered me at the University of Salford, Manchester. I also thank my supervisor in this intern-ship, Farid Meziane, for his contribution to my work and for receiving me in Salford as I was one of his own team.

I warmly thank ISEP for receiving me, specially to every member of the team RDI. I want to give special thank Sylvain Lefebvre for his help, his support, our long talks and his friendship, and Maria Trocan for taking care of me.

I thank my family. Thank my father for being a model of cleverness and intelligence. Thank my mother for learning me the affection. Thank my brother for reminding me that hard-work and achievements come together. My self-exigence often blinds me and my family always remind me who I am and what I am capable of.

Thank my other family, my friends. Those who are far-away and those who are near-by. Sathiya, Amadou, Rayane, Loïc and Denis, thank you for the daily smiles we shared together and your unconditional help whenever it was needed (or not). One day, Zakia told me that "phd collegues are friends for the whole life", and I already started missing you. I want to thank my love, Virginia, for supporting me anytime and helping me to put the last and hard period to this thesis.

Finally, thank FIORA project, "DGCIS" and "Conseil Regional de l'Île de France" for funding this thesis.

Abstract

Recommender Systems aim at pre-selecting and presenting first the information in which users might be interested. This has raised the attention of the e-commerce, where the interests of users are analysed in order to predict future interests and to personalize the offers (a.k.a. items). Recommender systems exploit the current preferences of users and the features of items/users in order to predict their future preference in items.

Although they demonstrate accuracy in many domains, these systems still face great challenges for both academia and industry; they require distributed techniques to deal with a huge volume of data, they aim to exploit very heterogeneous data, and they suffer from cold-start, situation in which the system has not (enough) information about (new) users/items to provide accurate recommendations.

Among popular techniques, Matrix Factorization has demonstrated high accurate predictions and scalability to parallelize the analysis among multiple machines. However, it has two main drawbacks: (1) difficulty of integrating external heterogeneous data such as items' features, and (2) the cold-start issue.

The objective of this thesis is to answer to many challenges in the field of recommender systems: (1) recommendation techniques deal with complex analysis and a huge volume of data; in order to alleviate the time consumption of analysis, these techniques need to parallelize the process among multiple machines, (2) collaborative filtering techniques do not naturally take into account the items' descriptions in the recommendation, although this information may help to perform more accurate recommendations, (3) users' and items' descriptions in very large dataset contexts can become large and memory-consuming; this makes data analysis more complex, and (4) the new user cold-start is particularly important

to perform new users' recommendations and to assure new users fidelity.

Our contributions to this area are given by four aspects: (1) we improve the distribution of a matrix factorization recommendation algorithm in order to achieve better scalability, (2) we enhance recommendations performed by matrix factorization by studying the implicit interest of the users in the attributes of the items, (3) we propose an accurate and low-space binary vector based on Bloom Filters for representing users/items through a high quantity of features in low memory-consumption, and (4) we cope with the new user cold-start in collaborative filtering by using active learning techniques.

The experimentation phase uses the publicly available MovieLens dataset and IMDb database, what allows to perform fair comparisons to the state of the art. Our contributions demonstrate their performance in terms of accuracy and efficiency.

Résumé

Les systèmes de recommandation visent à présélectionner et présenter en premier les informations susceptibles d'intéresser les utilisateurs. Ceci a suscité l'attention du commerce électronique, où l'historique des achats des utilisateurs sont analysés pour prédire leurs intérêts futurs et pouvoir personnaliser les offres ou produits (appelés aussi items) qui leur sont proposés. Dans ce cadre, les systèmes de recommandation exploitent les préférences des utilisateurs et les caractéristiques des produits et des utilisateurs pour prédire leurs préférences pour des futurs items.

Bien qu'ils aient démontré leur precision, ces systèmes font toujours face à de grands défis tant pour le monde académique que pour l'industrie: ces techniques traitent un grand volume de données qui exige une parallélisation des traitements, les données peuvent être également très hétérogènes, et les systèmes de recommandation souffrent du démarrage à froid, situation dans laquelle le système n'a pas (ou pas assez) d'informations sur (les nouveaux) utilisateurs/items pour proposer des recommandations précises.

La technique de factorisation matricielle a démontré une précision dans les prédictions et une simplicité de passage à l'échelle. Cependant, cette approche a deux inconvénients: la complexité d'intégrer des données hétérogènes externes (telles que les caractéristiques des items) et le démarrage à froid pour un nouvel utilisateur.

Cette thèse a pour objectif de proposer un système offrant une précision dans les recommandations, un passage à l'échelle pour traiter des données volumineuses, et permettant d'intégrer des données variées sans remettre en question l'indépendance du système par rapport au domaine d'application. De plus, le système doit faire face au démarrage à froid utilisateurs car il est important de fidéliser et satisfaire les nouveaux utilisateurs.

Cette thèse présente quatre contributions au domaine des systèmes de recommandation: (1) nous proposons une implémentation d'un algorithme de recommandation de factorisation matricielle parallélisable pour assurer un meilleur passage à l'échelle, (2) nous améliorons la précision des recommandations en prenant en compte l'intérêt implicite des utilisateurs dans les attributs des items, (3) nous proposons une représentation compacte des caractéristiques des utilisateurs/items basée sur les filtres de bloom permettant de réduire la quantité de mémoire utile (4) nous faisons face au démarrage à froid d'un nouvel utilisateur en utilisant des techniques d'apprentissage actif.

La phase d'expérimentation utilise le jeu de données MovieLens et la base de données IMDb publiquement disponibles, ce qui permet d'effectuer des comparaisons avec des techniques existantes dans l'état de l'art. Ces expérimentations ont démontré la précision et l'efficacité de nos approches.

Résumé étendu

0.1 Introduction

Le contenu web est devenu très vaste et les utilisateurs rencontrent des difficultés pour trouver une information pertinente. Les utilisateurs d'Internet lisent environ 10 Mo d'information par jour (par exemple, les actualités), entendent 400 Mo par jour (par exemple, musique) et voient, en géneral, 1 Mo d'information par seconde (par exemple, des vidéos)[1]. Ils peuvent se sentir débordés en cherchant parmi ce contenu ceux qui sont les plus adaptés à leur profil parce qu'ils n'ont pas le temps ou simplement ils n'ont pas la connaissance pour trouver le plus approprié.

Les systèmes de recommandation réduisent radicalement la quantité d'information présentée aux utilisateurs. Ils analysent leur intêret et présentent d'abord les items (un produit ou un service tels qu' une vidéo, une image, etc) qui les intéresseraient le plus [Kantor et al., 2011].

L'intérêt des utilisateurs pour quelques items peut être donné de façon explicite ou implicite appelé également "score". D'une part, l'intérêt explicite, comme une note ou un "rating", est une déclaration formelle de l'intérêt de l'utilisateur. Ces données sont le plus utilisées dans la littératuree. Les utilisateurs peuvent noter des items (par exemple des films) utilisant une note entre 1 et 5, où "1" représente le manque d'intérêt et "5" représente un fort intérêt pour l'item. D'autre part, l'intérêt implicite est une déduction donnée par l'interaction web entre des utilisateurs et des items [Oard et al., 1998]. En effet, il est possible de mesurer les comportements des utilisateurs par les pages Web. Ceci permet de connaître combien de clics un item a reçu ou combien de temps un utilisateur

[1] http://www.economist.com/node/8312260

a passé à consulter un item. De plus, les systèmes de recommandation peuvent incorporer les descriptions des items (par exemple le genre d'un film, le chanteur des chansons, les écrivains des livres), les données démographiques des utilisateurs (par exemple l'âge, le sexe, le niveau d'études) et des données venant d'autres contextes, comme Wikipédia, des réseaux sociaux, des coordonées gps, etc, ce qui peut aussi améliorer les recommandations.

0.1.1 Challenges

Les systèmes de recommandation font face à de multiples problématiques tels que la volumétrie et analyse de la donnée, l'hétérogénéité de la donnée, ainsi que la précision de recommandations. Ces défis sont résumés par la suite:

Adaptabilité et passage à l'échelle: le grand nombre d'utilisateurs, d'items et de scores supposent l'analyse d'une grande volumétrie de données, ce qui nécessite beaucoup de ressources et de temps. Il est nécessaire de créer des techniques de recommandation où le calcul est facilement parallélisable et incremental[Sarwar et al., 2002, Koren, 2010, Owen et al., 2011].

Hétérogénéité de la donnée: les systèmes de recommandation peuvent utiliser des informations hétérogènes externes autres que les intérêts des utilisateurs. Ces systèmes utilisent typiquement des ressources d'information multiples pour améliorer les recommandations [Kantor et al., 2011], par exemple les descriptions des items, les descriptions des utilisateurs, etc. Le défi de ces systèmes est de récupérer, classifier et intégrer ces informations, ce qui implique davantage de complexité, et affecte les performances du système.

Démarrage à froid: c'est la situation dans laquelle le système n'a pas (ou pas assez) d'informations sur un nouvel utilisateur/item, c'est-à-dire leur score; et par conséquence, les recommandations de l'utilisateur (ou de l'item) ne sont pas pertinentes [Su and Khoshgoftaar, 2009, Kantor et al., 2011].

L'objectif de cette thèse est d'améliorer les systèmes de recommandations en répondant aux défis cités précédemment.

0.2 État de l'art

Les systèmes de recommandation se doivent de sélectionner les informations les plus intéressantes en fonction du but recherché, tout en conciliant nouveauté, surprise et pertinence. Un système de recommandation se base sur des caractéristiques de références acquises de manière automatisée selon des méthodes différentes. Elles peuvent provenir de:

- L'item (l'objet à recommander) lui-même, on parle alors "d'approche basée sur le contenu" (ou content-based approach) [Balabanović and Shoham, 1997]. Le filtrage basé sur le contenu calcule la similarité entre les objets afin de trouver l'objet correspondant le plus aux goûts de l'utilisateur. Dans ce cas, l'utilisateur se voit recommander des items similaires à ceux qu'il a préférés dans le passé.

- L'utilisateur et l'environnement social, on parle alors "d'approche de filtrage collaboratif" (ou collaborative filtering). Le principe du filtrage collaboratif [Breese et al., 1998] consiste à implanter informatiquement le principe du "bouche-à-oreille". Il utilise les comportements connus d'une population pour prévoir les futurs agissements d'un individu. La méthode collaborative présente des avantages par rapport au filtrage basé sur le contenu: elle est plus efficace dans la pratique et simple à mettre en oeuvre. Notamment, il a été prouvé que les techniques de factorisation de matrice fournissent des résultats précis et ont l'avantage d'être facilement parallélisable (pour la montée en charge et le passage à l'échelle) [Koren et al., 2009].

Les milieux académiques et l'industrie ont investi leurs efforts dans le champ de la recommandation. Plusieurs études ont montré que la technique de factorisation matricielle est l'une des plus importantes contributions. Elle est apparue pendant le prix de "Netflix" et elle a aidé les nouveaux systèmes à améliorer la précision et le passage à l'échelle des techniques de recommandation. Cette technique n'utilise pas les descriptions, donc n'aide pas à l'incorporation de données hétérogènes. De plus, elle est très sensible aux situations de démarrage à froid et "sparsité". En conséquence, plusieurs travaux dans le milieu industriel mais aussi académique la combinent avec d'autres approches, notamment des approches utilisant le contenu des items, pour améliorer la performance globale des recommandations.

Cette thèse assume la factorisation matricielle comme la technique de principe dans les systèmes de recommandation et vise à aller au-delà de cette technique. Nos contributions se concentrent (1) sur le passage à l'échelle de la factorisation matricielle pour réaliser la meilleure performance dans des clusters Hadoop, (2) l'intérêt implicite d'utilisateurs dans les descriptions des items et comment réduire l'impact du domaine de recommandations, (3) les représentations des items et utilisateurs dans des très grands contextes de données et (4) le démarrage à froid orienté aux nouveaux utilisateurs pour des techniques de filtrage collaboratives pures. Notre but final est que ces contributions aident les systèmes de recommandation à mieux manipuler des données et comprennent mieux les utilisateurs pour améliorer la performance du système et la précision des recommandations.

0.3 Descente de gradient stochastique et distribuée

Le nombre d'utilisateurs et d'items dans un système de recommandation est habituellement très élevé. Par exemple, Netflix dispose de plus de 20 millions de clients, 80 milliers de films et 5 milliards de notes [Makari et al., 2014]. En général, les utilisateurs notent uniquement certains de ces items et cela crée un grand volume d'évaluations manquantes/inconnues pour prédire. Ainsi, ces prédictions peuvent affecter le temps ede traitement. Les nouveaux systèmes de recommandation devraient être précis dans leurs prédictions mais aussi parallélisables pour soulager la montée en charge et le temps de traitement des données.

Ce travail se concentre sur la technique de factorisation matricielle appelée la descente distribuée et stochastique de gradient (Distributed Stochastic Gradient Descent (DSGD)), contrairement aux technnique de optmisation par moindres carrés (Alternating Least Squares (ALS)[Zhou et al., 2008]). Les contributions dans ces aspects sont: (1) une mise en oeuvre de cette technique dans Hadoop/MapReduce, et (2) nous avons étudié cette technique pour améliorer son adaptabilité dans un ensemble de machines et ainsi améliorer la performance des recommandations.

0.3.1 Contribution et résultats

L'idée proposée dans [Gemulla et al., 2011, Makari et al., 2014] consiste à diviser une matrice initiale composée d'utilisateurs, d'items et de ratings en plusieurs blocs, de façon à ce qu'il y ait un certain nombre de blocs qui soient complètement indépendants (qui ne partagent pas d'utilisateurs ni d'items). Ces blocs indépendants forment un ensemble appelé "stratum". Lorsque les blocs sont indépendants, leur analyse est indépendante et leurs résultats sont aussi indépendants et agrégeables. Par conséquent, l'analyse des ratings (des blocs) dans le stratum peut être distribuée parmi plusieurs machines. Au maximum, autant de machines que de blocs dans le stratum sont éxéctuées en parallèle. En créant différents stratums (avec d'autres blocs indépéndants) il est possible de faire l'analyse de la matrice initiale complète.

Le problème que nous avons soulevé réside dans la décomposition de la matrice initiale en blocs. Imaginez une matrice de dimensions n_u x n_i, le point-clé dans la décomposition de bloc est la divisibilité des dimensions dans des parties d'un entier. Cela signifie que les dimensions pourraient être divisibles par b ("b" étant la quantité de machines disponibles à utiliser en parallèle) pour proprement créer des blocs ($n_u \% b = 0$ et $n_i \% b = 0$). Mais ceci n'est souvent pas le cas.

Dans [Gemulla et al., 2011, Makari et al., 2014], l'effacement de rangées/colonnes (c'est-à-dire des utilisateurs/items) résout cette condition de divisibilité. Cependant, ceci peut créer une perte d'exactitude: si le nombre de données supprimées est élevé, le système peut perdre des informations importantes. En fait, cela représente un compromis entre précision et flexibilité. Au contraire, l'ajout de dimensions peut augmenter le temps d'exécution de l'algorithme. Cependant, ceci n'affecte pas la précision, vu que l'insertion de la dimension vide n'ajoute pas d'information.

En utilisant ces idées, nous avons développé un aspect plus flexible dans la distribution. Nous proposons trois modes de décomposition de blocs qui affectent la création des "stratums": (1) sous-dimension, (2) dimension supérieure et (3) dimension flexible. Dans ce contexte, une dimension est un utilisateur (dimension de rangées) ou un item (dimension de colonnes). La sous-dimension est la proposition donnée par [Makari et al., 2014]. Ce

mode redimensionne la matrice en supprimant des dimensions. La dimension supérieure redimensionne la matrice en insérant des dimensions vides jusqu'à l'accomplissement de la condition de décomposition. Finalement, la technique de dimension flexible est un hybride de ces deux techniques. Elle cherche la dimension la plus proche (dessus ou dessous) des dimensions réelles.

Nous évaluons la performance de nos propositions en utilisant le jeu des données publiques de MovieLens[2], qui a 10 millions d'évaluations appliquées par 69878 utilisateurs sur 10667 films. Nous créons une matrice de rangées et des colonnes avec les notes des utilisateurs aux items. L'évaluation et les comparaisons sont effectuées en termes de précision et d'adaptabilité au nombre de machines dans le cluster.

Nos techniques donnent un meilleur résultat en termes de précision lorsqu'elles évitent l'effacement de données (utilisateurs, items et ratings). Le Tableau 1 présente une comparaison de nos résultats avec des autres techiniques de factorisation matricielle, où la mesure de précision utilisée est le erreur moyenne quadratique (RMSE). Nous gagnons environ 1% d'amélioration en exploitant les machines dans le cluster (quantité de noueds b).

0.4 L'intérêt implicite des utilisateurs dans les attributs des items

Les technologies de recommandation font aujourd'hui face à des défis scientifiques majeurs: comment intégrer l'hétérogénéité des sources d'information pour modéliser les préférences, comment découvrir des nouvelles préférences, comment traiter efficacement ces masses d'information, quels types d'interfaces faut-il considérer?

Par ailleurs, les approches filtrage collaboratif et basé sur le contenu citées précédemment présentent des inconvénients principalement liés à l'hétérogénéité des sources d'information et à la montée en charge du système d'où la nécessité de mettre en place des algorithmes performants et robustes. Ceci est l'objectif de cette étude en vue d'améliorer la qualité des systèmes de recommandation en introduisant de la "sémantique" aux données et en

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

Table 1: Résultats de nos expérimentations en termes de précision et quantité de noueds utilisés.

Technique	RMSE	Quantité de noueds (b)
ALS	0.79603	1
Sous-dimension DSGD	0.77571	1
Dimension flexible DSGD	0.77571	1
Dimension supérieure DSGD	0.77571	1
Sous-dimension DSGD	0.77611	2
Dimension flexible DSGD	**0.77555**	2
Dimension supérieure DSGD	0.77559	2
Sous-dimension DSGD	0.77626	5
Dimension flexible DSGD	0.77617	5
Dimension supérieure DSGD	**0.77597**	5
Sous-dimension DSGD	0.77586	7
Dimension flexible DSGD	**0.77548**	7
Dimension supérieure DSGD	0.77565	7
Sous-dimension DSGD	0.77593	15
Dimension flexible DSGD	0.77596	15
Dimension supérieure DSGD	**0.77555**	15

distribuant les traitements afin de minimiser les temps de calcul. La "sémantique" est ici définie comme une extension dans le sens de la donnée courante, les scores (ratings), afin de créer des "ratings sémantiques".

0.4.1 Contribution et résultats

Afin de fournir une généricité dans le domaine d'application, un passage à l'échelle et une recommandation précise, nous proposons un système à trois couches: une couche de pré-analyse, une couche "sémantique" et une couche de recommandation.

Dans une première étape, nous nous intéressons aux préférences des utilisateurs pour le contenu des items. Cette information est souvent manquante lorsque ce contenu peut être très large et varié, et donc difficile d'attirer d'information de score explicite. Nous proposons d'étudier l'intérêt implicite des utilisateurs dans le contenu des items en étudiant les préférences passées sur les items. Ceci est fait à travers un module de comptage qui met en lien chaque utilisateur avec le contenu des items. Lorsqu'un contenu est répété (le même

acteur qui apparaît dans deux films différents), le compteur d'intérêt de cet utilisateur pour ce contenu est incrémenté aussi.

La couche sémantique exploite cette nouvelle information ainsi que le rating données par les utilisateurs pour les items. Ceci se traduit par la transformation sémantique des notes des utilisateurs. Nous nous intéressons tout d'abord au nombre d'occurrence des attributs qui ont été notés par un utilisateur. Nous appelons cette occurrence « la fréquence d'apparition » ou « coïncidence » : cette valeur correspond au nombre de fois que les valeurs des attributs se répètent dans les items notés par l'utilisateur. Cette valeur est extraite à partir des compteurs déjà précalculés. La transformation du rating consiste en l'addition de ce nouvel intérêt implicite à l'intérêt explicite, à travers l'équation suivante:

$$sv_{u,i} = r_{u,i} + E[r_{u,*}] * \frac{\left|\sum_{j=1}^{F} C_j * W_j\right|}{N_u} \tag{1}$$

Avec F le nombre total des attributs, N_u le nombre total des items notés par l'utilisateur "u". C_j est la fréquence d'apparition de l'attribut j dans l'ensemble des items qui ont été notés par l'utilisateur et W_j étant un poids calculé à partir d'une phase de sélection des attributs par une analyse des composantes principales. $E[r_{u,*}]$ est la moyenne des notes de l'utilisateur et $r_{u,i}$ est la valeur du rating initial donnée à l'item "i".

L'équation sémantique peut être appliquée à deux niveaux dans la recommandation. D'une part, nous pouvons appliquer cette équation à toutes les notes disponibles dans la base de données initiale, ce qui permet de mieux expliquer l'intérêt des utilisateurs pour les caractéristiques définissant les items notés (ajouter du sens à la note). D'autre part, nous pouvons faire le choix d'appliquer l'équation sémantique à la sortie de la recommandation. Supposons que le module de recommandation renvoie un résultat des top K items (les K items les plus pertinents) pour un utilisateur donné, avec une estimation de la note pour ces top K. Ces notes seront transformées en une note sémantique suivant l'équation précédente et les items proposés seront réordonnés en conséquence en top K', K' pouvant être inférieur ou égal à K.

Enfin, le module de recommandation utilise une technique de filtrage collaboratif basée sur une méthode de factorisation de la matrice pour générer des recommandations précises.

En effet, cette technique a montré son efficacité comme méthode de filtrage collaboratif pour la recommandation [Koren and Bell, 2011].

Dans la phase d'expérimentation, nous comparons nos approches (appliquer la "sémantique" aux ratings initiaux, ou bien aux prédictions dans le top-K) et la technique de factorisation matricielle de base sans l'utilisation de l'équation sémantique. Nous utilisons les métriques de précision et rappel, parce que ces techniques n'évaluent pas la qualité de la prédiction des notes, mais la pertinence des items qui sont proposés aux utilisateurs. La précision calcule la probabilité qu'un item pertinent soit choisi et le rappel étant la probabilité qu'un item choisi soit pertinent. Nous utilisons aussi la F-mesure pour combiner le rappel et la précision dans une seule métrique afin de faciliter la comparaison. Nos approches donnent de meilleurs résultats que la technique de matrice de factorisation sans sémantique.

De plus, nous nous sommes interessés à l'impact de nouveaux ratings sémantiques sur la similarité des items à recommander. Nous utilisons la métrique d'ILS (*Intra-List Similarity*), appelée également ILD (*Intra-List Diversity*) pour mesurer la diversité/similarité entre les items dans la liste des top-K présentée à l'utilisateur.

Un bon système de recommandation doit trouver l'équilibre entre ces deux concepts diversité et similarité. En effet, des items trop diversifiés peuvent provoquer une confusion chez l'utilisateur, alors que recommander toujours les mêmes items peut ennuyer celui-ci. La matrice de factorisation a tendance à faire de recommandation diversifiées. Notre approche permet, dans cette diversité, de retourner des items plus similaires dans le top-K. Ceci est du au fait que nous prenons en compte l'intérêt pour les attributs afin d'identifier les items susceptibles d'intéresser l'utilisateur.

0.5 Modèle de similarité avec des filtres de Bloom

La description des items et d'utilisateurs peut être très précise puisqu'un grand nombre de caractéristiques peut être utilisé (par exemple des acteurs, des directeurs, des auteurs). On peut même imaginer l'incorporation de données externes telles que les données ouvertes ou des données provenant de réseaux sociaux, etc. pour mieux décrire les items/les utilisa-

teurs [Kantor et al., 2011, Peis et al., 2008, Dahimene et al., 2014]. Or ceci provoque une description certes détaillée mais gourmande en terme d'espace de stockage ou d'utilisation mémoire. Ce problème est particulièrement présent dans le cadre des méthodes de filtrage à base de contenu et des méthodes hybrides.

Souvent, ces techniques utilisent une représentation vectorielle de l'item ou utilisateur pour décrire ses caractéristiques et établir des comparaisons entre les items/utilisateurs. Ce nombre croissant de caractéristiques a trois conséquences importantes : (1) la grande taille des vecteurs, (2) les données de plus en plus éparses dans ces vecteurs et (3) le temps mis pour exécuter des opérations sur ces vecteurs. De plus, la quantité de caractéristiques change la similitude, et généralement plus d'attributs utilisés augmentent la qualité de la similitude. Dans cette représentation vectorielle, ajouter des caractéristiques ajoute de nouvelles dimensions au vecteur. De plus, les techniques de sélection des caractéristiques et la réduction spatiale peuvent induire une perte de précision dans la représentation des items ainsi que dans le calcul de similarité entre les items.

Dans ce contexte, nous proposons une technique de représentation des données qui crée un vecteur compressé en réduisant la taille totale d'un vecteur réel, et nous l'utilisons aussi pour développer tout un système capable d'exploiter cette représentation pour exécuter des opérations de similarité entre les items/utilisateurs. Ceci permet de comparer une large quantité de caractéristiques qui décrivent les items/utilisateurs tout en gardant une taille de vecteur très réduite.

0.5.1 Contribution et résultats

Nous proposons d'utiliser des représentations de filtre de bloom. Un filtre de bloom est une structure de bit qui représente "n"-éléments d'un même ensemble "S" dans un espace plus réduit de "m"-bits [Broder and Mitzenmacher, 2004]. Initialement, les m-bits sont mis à "0" ce qui représente l'absence des éléments insérés dans le filtre. Puis, on utilise "k" fonctions de hachage pour distribuer efficacement l'insertion des éléments: pour insérer un élément, celui-ci est "haché", ce qui retourne "k" positions de la structure de bits à passer à "1".

La motivation et les principales contributions sont:

1. Les filtres de bloom réduisent fortement la taille des représentations des items tout en gardant le même nombre de caractéristiques. Cela peut induire l'augmentation de caractéristiques utilisées pour décrire les items/utilisateurs;

2. Les filtres de bloom autorisent des opérations binaires rapides, comme l'intersection "AND" qui permet de comparer les caractéristiques communes des items, et permet de mesurer la similarité. La Figure 1 montre un exemple de cette opération avec les filtres de bloom et une représentation d'un point de vue d'ensemble de cette opération;

3. Dans la recherche des similarité entre les items, nous trouvons des caractéristiques communes, mais il est aussi possible d'avoir des caractéristiques manquantes communes. Pour y faire face, nous proposons une autre opération binaire, le "XNOR", qui prend en compte autant des caractéristiques communes que des caractéristiques manquantes communes. Par exemple, l'intersection prend en compte les éléments communs-insérés de deux ensembles. Etant donné les caractéristiques $x1$, $x2$, $x3$, $x4$ et deux ensembles (items) $S_A = x1, x3$ et $S_B = x2, x3$, l'intersection des deux ensembles est $S_C = x3$. Cependant, les ensembles S_A et S_B ont plus en commun que cette intersection. En réalité, l'élément $x4$ n'est dans aucun de ces ensembles, et par conséquent, $x4$ est commun à S_A et S_B. La Figure 2 montre un exemple de cette opération avec les filtres de bloom et une représentation d'un point de vue d'ensemble de cette opération;

Nous expérimentons nos approches sur le jeu de données publiques de MovieLens [Cantador et al., 2011] et nous comparons le résultat avec des approches de représentation vectorielle et de similarité cosinus/jaccard. Nous démontrons que la représentation de filtre de bloom réduit fortement la taille des représentations vectorielles (environ 94-97% par vecteur), tout en gardant une haute fidélité lorsque des opérations de similarité sont effectuées (environ 98% de précision).

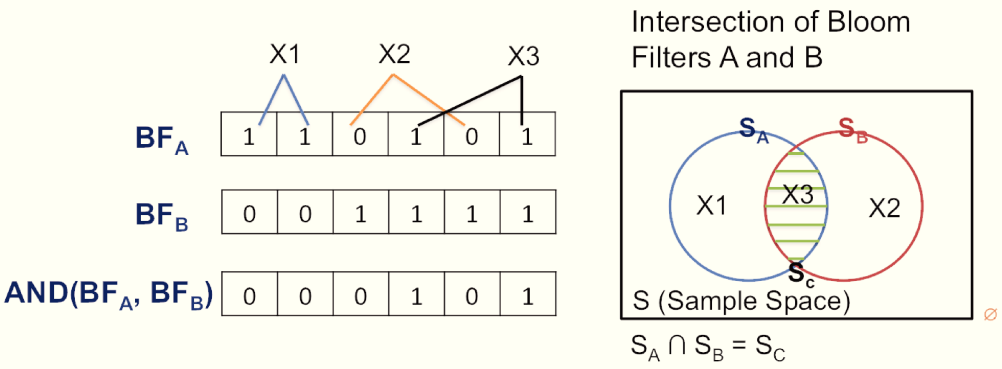


Figure 1: Représentation et exemple de la simitlarité "AND" en utilisant un modèle de filtres de bloom.

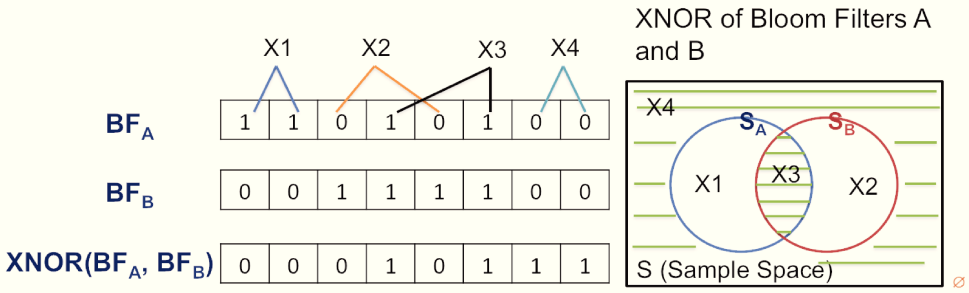


Figure 2: Représentation et exemple de la simitlarité "XNOR" en utilisant un modèle de filtres de bloom.

0.6 Apprentissage actif pour les systèmes de recommandation

Les systèmes de recommandation ont démontré une grande précision en prédisant les intérêts des utilisateurs "connus", c'est-à-dire des utilisateurs dont on connaît certains intérêts. Cependant, ces systèmes souffrent toujours du démarrage à froid de l'utilisateur, où le système n'a pas d'informations sur le nouvel utilisateur et les recommandations précises/personnalisées sont presque impossibles [Su and Khoshgoftaar, 2009]. Le démarrage à froid est particulièrement fort dans des recommandations de filtrage collaboratives parce que ces systèmes se basent seulement sur les notes (i.e. ratings) d'utilisateurs [Golbandi et al., 2011].

Une technique pour faire face à ce problème est l'apprentissage actif pour des systèmes de recommandation, une méthode d'acquisition de données pour des situations de démar-

rage à froid. L'apprentissage actif propose l'interaction d'utilisateurs avec le système pour obtenir des données (ratings) de grande qualité [Elahi et al., 2014]. Un questionnaire simple est utilisé pour obtenir les premiers ratings rapidement, par exemple: aimez-vous ce film? pouvez-vous noter ce livre?. Donc, on demande aux utilisateurs de répondre à ces questions en donnant leur avis, de cliquer par exemple sur "j'aime", "je n'aime pas" ou "je ne connais pas".

Cependant, les utilisateurs ne souhaitent pas évaluer beaucoup d'items et poser beaucoup de question car cela peut supposer une perte de temps pour eux [Harpale and Yang, 2008, Karimi et al., 2011a]. En conséquence, les questionnaires devraient être courts et rapides dans l'analyse. Il faudra donc poser des questions soigneusement choisies pour récuperer le plus d'information des utilisateurs [Zhou et al., 2011, Karimi et al., 2011a].

Nous proposons une technique d'apprentissage actif basée sur le concept des arbres de décision. Le but des arbres de décision dans l'apprentissage actif est de diviser la population d'utilisateurs selon leurs préférences sur quelques produits.

Ainsi, des utilisateurs dans le noeud de l'arbre ont tendance à partager des préférences semblables. En descendant dans l'arbre, ces groupes sont raffinés et les préférences sont mieux détectées. Ces techniques, dans un contexte de filtrage collaboratif, ont seulement accès aux notes des utilisateurs sur les items, i.e. ratings. Par conséquent, ces techniques ont pour but de découvrir des items dans chaque noeud de l'arbre, qui séparent efficacement la population des utilisateurs selon leurs ratings pour ces items.

Cette idée est très utile pour améliorer la performance du système de recommandation dans un contexte de démarrage à froid. Un noeud est représenté par une question (l'item), par exemple "aimez-vous ce film?". Le nouvel utilisateur peut donc répondre à ces questions. En donnant la réponse, le système identifie mieux les préférences de cet utilisateur et peut trouver une meilleure nouvelle question à lui poser en utilisant l'arbre de décision.

0.6.1 Contribution et résultats

Les techniques d'arbres de décision actuelles exploitent seulement les ratings existant dans R pour (1) la découverte de questions à poser, (2) la segmentation de la population

à partir de questions et (3) le calcul de prédiction de ratings que le nouvel utilisateur pourrait donner (la prédiction de la réponse). Le problème est donc, à chaque noeud de l'arbre de décision, de trouver un item qui permettra de mieux identifier les préférences d'un nouvel utilisateur. Ceci est un problème d'optimisation qui se sert des ratings existant dans chaque noeud de l'arbre pour trouver le meilleur item à proposer comme question.

Cette technique utilisée dans l'état de l'art est simple pour que l'arbre de décision soit rapide à construire. Elle utilise uniquement les ratings des utilisateurs pour prédire la moyenne et ainsi évaluer la pertinence à poser cet item comme question.

Nous proposons d'utiliser non seulement le rating r d'un utilisateur u pour un item j, mais aussi une prédiction p de rating d'un utilisateur u pour un item j, de sorte qu'il y ait toujours un rating $r_{u,j}$ associé à une prédiction $p_{u,j}$.

Nous proposons d'utiliser des évaluations $r_{u,j}$ uniquement pour diviser la population d'utilisateurs. Les prédictions $p_{u,j}$ seront ainsi utilisées pour découvrir des items à poser comme question.

Notre approche est implementée en utilisant une technique d'arbres de décision non supervisée et une autre technique supervisée. Nous comparons les deux approches avec l'état de l'art existant. Nos approches sont capables de trouver des items plus pertinents que les items trouvés par les approches de l'état de l'art. Ainsi, nos approches identifient les préférences des nouveaux utilisateurs plus rapidement (en un moindre nombre de questions).

0.7 Conclusions et Perspectives

Les systèmes de recommandation sélectionne, parmi une grande quantité de données, les informations pour lesquelles les utilisateurs pourraient être les plus intéressés. Ils exploitent les préférences connues d'utilisateurs dans des items, qui sont donnés de façon explicite ou le retour d'information implicite d'utilisateurs, pour prévoir les préférences futures d'utilisateurs dans d'autres items (inconnus). De plus, il est possible pour certaines techniques de recommandation d'utiliser la description d'items ou d'autres données hétérogènes afin d'améliorer ces recommandations.

Typiquement ces techniques sont classifiées par: filtrage à base de contenu, filtrage collaboratif et méthodes hybrides. Cependant, la technique la plus utilisée est de nos jours une technique de filtrage collaborative appelée la mactorisation matricielle. Cette technique a démontré une grande pécision et une grande facilité pour traiter la montée en charge et le passage à l'échelle. Cependant, elle ne facilite pas l'utilisation de données hétérogènes. De plus, elle souffre du démarrage à froid.

Nos travaux de recherche se concentrent sur plusieurs aspects différents que tout système de recommandation devrait prendre en compte: (1) la qualité et la précision des recommandations, (2) la représentation d'items/utilisateurs, (3) la distribution et l'adaptabilité du système, et (4) le démarrage à froid. Nos contributions ont fait face à ces problèmes:

- L'amélioration de techniques de filtrage collaboratives en ajoutant une couche externe pour analyser l'intérêt implicite des utilisateurs dans l'attribut des items. Cette méthode réduit l'impact de la dépendance de domaine dans des recommandations et améliore la qualité des recommandations. En fait, nous croyons que les recommandations d'items qui contiennent des caractéristiques plus connues pour l'utilisateur (par exemple les acteurs, des réalisateurs) vont plus probablement être acceptables.

- Particulièrement dans des systèmes à base de contenu et hybrides, traiter la grande quantité de données de caractéristiques peut devenir problématique. La sélection de caractéristiques et des techniques de réduction spatiales sont utilisés pour faire face à ce problème, pourtant ils cèdent inévitablement à une perte d'informations. En effet, la qualité de la représentation d'utilisateurs et des items est affectée tout comme les opérations de similarité parmi les items. Nous avons proposé de représenter des items/utilisateurs dans des structures spatiales de très basse consomation de mémoire appelées filtre de bloom. De plus, ceci nous permet de définir deux mesures de similitude: "AND", pour prendre en compte des caractéristiques communes, et "XNOR", pour prendre en compte des caractéristiques communes et des caractéristiques manquantes communes.

- Amélioration de la précision et de l'adaptabilité d'une technique de factorisation matricielle en proposant une approach distribuée basée sur MapReduce.

- Nous avons utilisé une technique d'apprentissage actif basé sur les arbres de décision pour faire face au problème de démarrage à froid dans le contexte d'un nouvel utilisateur. Cette technique permet d'améliorer les questionnaires présentés aux nouveaux utilisateurs en posant des questions plus pertinentes. De plus, elle permet de plus rapidement capter les préférences de cet utilisateur (une faible quantité de questions seraient posées).

Les perspectives qui se présentent sont variées. Nous voulons continuer ces travaux pour analyser et étudier les systèmes de recommandation dans une logique explicative (comment expliquer aux utilisateur la pertinence des recommandations proposées), de compromis d'exploration et d'exploitation (côté recherche et découverte des intérêts d'un utilisateur, versus, côté sur-exploitation de la donnée et monotonie dans les recommandations) dans un contexte de grand hétérogénéité de données, et des nouvelles techniques d'apprentissage actif pour le problème de démarrage à froid tel qu'une analyse du temps de réponse des utilisateurs aux questions posées dans le questionnaire.

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Motivation

"We are leaving the age of information and entering the age of recommendation"

- Chris Anderson in "The Long Tail"

1.1 Introduction

The web content has become so vast that users hardly find the information they are looking for. People read around 10 MB of data per day, hear 400 MB of audio per day, and see 1 MB of information every second [1]. In 2015, the media information consumption was estimated to 74 GB per consumer and per day [2]. Users may feel overwhelmed by this huge content because they may not have the time or simply they do not have the knowledge about what fits better to their needs.

Recommender Systems (RS) emerge from the Information Search and Retrieval (IR) field in order to cope with this information overload and decision-making issues. The information retrieval systems seek and filter information depending on users' queries [Baeza-Yates and Ribeiro-Neto, 1999]. These systems used to represent textual content (e.g. web pages and articles) as a set of words that are then matched with users' requests to return the relevant information for the user in a non user personalized experience; all users receive the same information for the same queries. Recommender Systems aim to cope with these issues by personalizing the content in webs for users [Kantor et al., 2011]. Indeed, they

[1] http://www.economist.com/node/8312260

[2] http://ucsdnews.ucsd.edu/pressrelease/u.s._media_consumption_to_rise_to_15.5_hours_a_day_per_person_by_2015

analyze the interaction of users with the information in webs in order to predict the future interactions that suit the users' interests. This allows to filter and organize items in order to present first the items of users' interest, i.e. recommendations.

Recommender systems dramatically reduce the amount of information presented to users and "delivers the correct information to the correct users" [Zhou et al., 2011]. The first recommender system, "Tapestry" [Goldberg et al., 1992], appeared in 1990 and it also worked looking for text-similarities and automatically learning from the users preferences. However, it did not contemplate non-text-similarities; it only analyzed text-content but not other contents such as videos, audio, etc. New techniques to afford this singularity emerged and the concept of "item" appeared to involve textual and non-textual information. Hence, it attracted the interest from many fields, such as e-commerce, service providers and media services, with the aim of personalizing the services and users' experience.

Nowadays, recommender systems are well studied in the academia and the industry. They are daily present almost everywhere giving suggestion of where to eat, which movie to watch, which articles to read, whose to be friend with, the publicity we receive, etc. The Internet is moving from the era of search to the era of recommendations; the searching allows to look for a specific information, whereas the recommendations allow to discover information you are not specifically looking for although it is of your interest. We stop looking for information, and information starts finding us.

As introduction to this thesis work, we first formally define recommender systems and present the recommendation actors and the recommendation process in Section 1.2. Section 1.3 presents the trend challenges for recommender systems. Section 1.4 presents our motivation and the main guidelines of our researches.

1.2 Definition and process of recommender systems

Recommender systems analyse the interests of users and present first the items in which they might be more interested [Kantor et al., 2011]. Their main purpose is to assist users in their daily decision-makings by presenting a reduced set of options in the form of recommendations or advices [Chee et al., 2001].

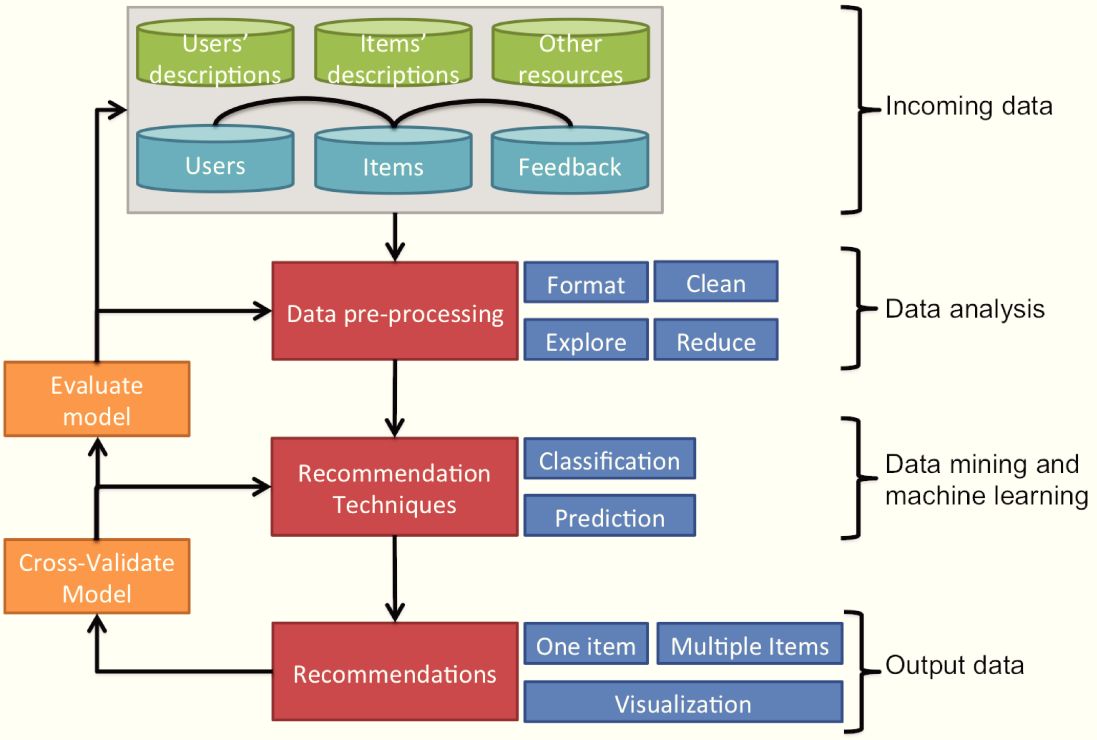
The users' interests represent the strength of their relations with items. They are also known as "score" or "feedback". On the one hand, the explicit feedback is a formal declaration of the user. The most typical explicit interests are the rating systems, where users can evaluate/rate items (e.g. movies) using a 0-5 stars scale. Other shorter and larger scales are possible, such as 'like, dislike, unknown', 0-10 stars, 0-20 stars. Another kind of explicit feedback may rely on comments and opinions. This feedback is usually processed using Natural Language Programming (NLP) to really understand the meaning of users' sentences, such as sentiment analysis. On the other hand, the implicit feedback is an interest deduction given by the interaction between users and items [Oard et al., 1998]. The so-called tracking systems measure the users' behaviors through the web-pages and collect users' examination (e.g. item selection), retention (e.g. save item, print page) and reference (e.g. share). This is based on the count of number of clicks, time spent on exploring an item, users' location, etc. It is worth to mention that recommender systems can use other kind of information to enhance the accuracy of recommendations [Kantor et al., 2011]. For instance, it is possible to incorporate the items' descriptions (e.g. the genre of a movie, the singer of songs, the author of books), the users' demographics (e.g. the age, the sex, the education) and data coming from other contexts, such as Wikipedia, social networks, gps-locations, etc.

Recommender systems are good to find a set of items in which the user can be interested, and to guess the interest (e.g. rating) that a user would have in a particular item. On the one hand, the term "user" is often used to define the (active) user for whom recommendations are currently required. On the contrary, the term "item" is very large and depends on the domain of the recommendation. For instance, recommender systems dealing only with movies would consider a movie as an item. Moreover, an item can be one user, such as the suggestions of a "friend" or a contact in social networks.

The users, items and feedbacks are the core-stone of the information used in recommender systems. This data is formatted, treated and analyzed by using techniques from data mining and machine learning fields in order to generate recommendations. Figure 1.1 shows the typical workflow representing the recommendation system process.

The incoming data is composed of, at least, the users' interests in items. Other data

Figure 1.1: Example of recommendation workflow.



resources are possible as well. The first step is to analyze the data and pre-process it and to format it to the algorithm's structure, to clean it by identifying anomalies, to explore it, and, eventually, to reduce it in order to evaluate the techniques or to avoid noisy data. The second step uses data mining and machine learning techniques to understand users' interests. The goal in this step is, roughly, to classify and organize users and items regarding the known feedbacks in order to predict to which group of interests belongs a given user/item. The third step is to present the recommendations to users. This whole process needs to be evaluated and validated to find the best possible model for generating recommendations.

1.3 Challenges in Recommender Systems

Recommender systems face divers challenges which affect their recommendation workflow. For instance, the incoming data would need to be clean before being processed by recommendation techniques, such as identifying shilling attacks (fake users profiles and

fake ratings to promote or degrade items)[Lam and Riedl, 2004]. Moreover, how many recommendations to present to users can be also a challenge: a few items may not be enough for the users' needs, and too many items may be too much. Where to place these recommendation to optimize the site-look and users' experience is also an interesting challenge.

In this section we consider the most simplistic representation of a recommender system process which assumes to have prepared incoming data ready to be analyzed by recommendation techniques in order to produce recommendation to users. As a consequence, we mainly focus on the challenges related to the recommendation techniques.

Scalability

The number of users, items and feedback can increase rapidly. Recommendation techniques may become computationally expensive and very time-consuming. It is necessary to parallelize the computation of these techniques among several processors or machines and to create easily incremental models in order to receive new data [Sarwar et al., 2002, Koren, 2010, Owen et al., 2011].

Data heterogeneity

Recommender systems can use external heterogeneous information in addition to users' interests in items. These systems typically use multiple information resources to enhance the recommendations to users [Kantor et al., 2011], e.g. the items' descriptions, users' keywords for items, users' descriptions, etc. The challenge of these systems is to retrieve, classify and integrate this big, heterogeneous and diverse information to the sake of recommendations, what usually induce more complexity and has a negative impact on the scalability of the system.

Similarity/Diversity

Recommendations to one user should not be always similar (e.g. comedy movies), otherwise the user could get bored. On the contrary if recommendations are too diverse, it can generate an untrusted feeling in the user. This is part of a similarity/diversity trade-off that a recommender system should take into account [Konstan and Riedl, 2012].

Sparsity

This often happens when the number of items is much larger than the number of users; and thus it is very complicated to have feedback for all items, e.g. there are millions of books in Amazon but a user may have bought hundreds of books. This situation generates poor informative datasets [Su and Khoshgoftaar, 2009, Kantor et al., 2011].

Cold start

The cold-start is the situation in which the recommender system has no or not enough information about the (new) users/items, i.e. their ratings/feedback; hence, the recommendation to users (or of items) are not well performed [Su and Khoshgoftaar, 2009, Kantor et al., 2011]. Particularly, the new user cold-start is the lack of users' preferences and the new item cold-start is the lack of ratings for this item.

Time-aware recommendations

Recommendation techniques exploit the current known users' interest in items. These may be linked to timestamps and thus it is possible to identify long time distant preferences as well as very recent ones. Generating proper recommendations by taking this information into account makes proposed recommendation to evolve accordingly to the users' preferences [Koren et al., 2009].

Multi-type and multi-criteria recommendations

On the one hand, multi-type recommendations aim to recommend not only the item itself, but also relevant information related to this item [Zhuo et al., 2011]. For instance, in a restaurant recommendation it is interesting to suggest the route to go to the restaurant. On the other hand, common recommender systems are considered single-criterion, because they only analyze the interest of users in items. Multi-criteria recommendations suggest that users' feedback are a result of combinations of several interests [Lakiotaki et al., 2008, Kantor et al., 2011]. For instance, the rating of a movie (considered as an item) may depend on precise aspects, such as the main story and the special effects. However,

the feedback for these other criterion are also required. Indeed, this method assumes that collecting more diversified data results in more accurate recommendations. In addition, multi-criteria systems may go further considering cross-domain recommendations, where ratings of users in one item's domain (e.g. movies) can be used for other completely different domains (e.g. restaurants) by looking for similarities in the interests of this user and other users in both domains (e.g. users who liked the same movies and they agree in one restaurant, may agree in other restaurants as well).

Exploration versus exploitation

In fact, recommender systems tend to be more accurate in predicting very popular items (i.e. items from which many ratings are known), and hence these items are more recommended as well. As a consequence, less rated items are little by little forgotten by the system. Under this context, the exploration paradigm consists in letting the user freely navigate among the items to discover and rate new unexpected items. On the contrary, the exploitation paradigm refers to the usage of the users' interest (explicit or implicit) and the persistence of recommendations, e.g. to continuously recommend the same items to users when these suggestions have not been taken in several occasions. How intrusive the recommender system may be in the life of users is an interesting exploration-exploitation trade-off to take into consideration [Balabanović, 1998, Rubens et al., 2011].

Users' experience: explanations and data privacy

The explanation of recommendations is an important factor for users decision-makings process [Tintarev and Masthoff, 2011]. The perception of quality and variety in recommendation is part of the users' experience. In some cases, this may create a lack of trustiness and confidence due to associated risks in accepting recommendations, e.g. to buy a book against to buy a car. In fact, users are not willing to risk much based on recommendations they do not understand [Konstan and Riedl, 2012]. The explanation of recommendations may alleviate this issue and it helps the users to make reasonable decisions [Herlocker et al., 2000].

In addition, the explanations come across three axis that recommender system should

accomplish [Bonhard and Sasse, 2006, Knijnenburg et al., 2012]: (1) transparency, to explain how the recommendation was performed, e.g. the users and/or items correlations involved; (2) trust, to encourage the user to accept recommendations; and (3) scrutability, which allows the user to interact with the system to communicate possible mistakes to avoid in the future. To deal with these axis is a complex challenge in recommender systems due to the heterogeneity of data and the representation of the information to explain (e.g. visualization). However, this challenge has to take the privacy of the user and his data into consideration.

Users' classification: grey sheep and black sheep

The grey sheep and black sheep are user classification issues where the system has not a clear understanding of some users' preferences because they are too diverse (grey sheep) or too specific (black sheep), what makes difficult to compare to other users' preferences [Su and Khoshgoftaar, 2009]. These challenges are complex to detect and sometimes are considered as acceptable drawbacks in recommender systems.

Other challenges

There are other challenges in recommender systems. For instance, there is a need to trust in the explicit ratings made by users. This data has a strong impact on recommendations, and hence more accurate rating lead to better recommendations. These challenges rely on the quality of the ratings. Some authors assume and accept this property of the data, although other authors aim to test and qualify this data [Amatriain et al., 2009], e.g. by asking users to re-rate a known item. Another example is the recommendation to/in groups of users/items. This challenge requires to satisfy several needs from users or to adapt items for users in order to create a unique but successful recommendation. The recommendation to groups deals with many users under the same current context. For instance, what kind of music to recommend in a party [Masthoff, 2011], or what TV programme suits to all members of the family. On the contrary, the recommendation in groups try to create mini set of items to propose together to the same user [Konstan and Riedl, 2012]. One example is the often offered tourism package of flight, hotel and rental

car set, or the bought-together package of Amazon.

1.4 Motivation

This chapter has introduced recommender systems and has given the first guidelines about the recommendation process and the challenges currently existing in this field.

Our main concern is the volume and variety that these systems face. On the one hand, the huge quantity of information (users, items, feedback and external data) makes the recommendation process analysis complex and time-consuming for one single machine. It is necessary that recommendation techniques become scalable and parallelizable among multiple machines to alleviate this issue. On the other hand, using heterogeneous data is not trivial in recommender systems. There are techniques that are explicitly designed to exploit it and others which simply assume it is not necessary. We believe that recommender system would evolve to improve the integration of all kind of data in order to achieve better accuracy and larger recommendation contexts. In addition, this heterogeneity together with the high volume of data induce to complex users and items representations. In fact, it is possible to highly detail users' and items' descriptions, however this data representation would become very difficult to deal with in terms of memory-space-consumption and exploitation of representations. Finally, there is a major challenge in recommender systems which is the cold start, particularly, the new user cold start. This typically occurs when new users sign up to the system and recommendations are required. As long as the recommender systems has not information about this user and/or his preferences, proper recommendation are nearly impossible.

Our role of this thesis is to answer to these challenges in terms of research and development. Chapter 2 presents the state of the art in the field of recommender systems. We first explain the assumptions, advantages and disadvantages of different recommendation techniques. Second, we give related works in the academic and industrial researches. In addition, recommender systems libraries are presented.

Chapter 3 deals with the scalability and parallelization of recommendation techniques. We analyze the matrix factorization techniques, which have demonstrated to be highly

parallelizable, and its scalability within a cluster of machines. Our goal is to enhance the parallelization of one particular matrix factorization technique called Stochastic Gradient Descent, which have demonstrated better accuracy in recommendations than other matrix factorization techniques but a more complex parallelization.

Chapter 4 focuses on the integration of heterogeneous data into the recommendation process. Our goal is that, collaborative filtering techniques, such as the matrix factorization, may exploit the items' description as well in the benefit of users' recommendations. We propose extract a new source of knowledge from past users' interests by using the items' descriptions: we analyze the interests of users in the description of items and not only in the item itself. This new information is added to the current interests of users in item and used by collaborative filtering approaches to compute more accurate recommendation for users.

In Chapter 5 we discuss that the integration of heterogeneous data to describe users and items may have an effect in the accuracy of such descriptions. Typically, the more data the system uses the better the users and items are detailed. However, this could lead to very long and difficult to handle descriptions. Our goal in this context is to create a low-memory space model that takes into account the high detailed descriptions and makes easy to compare users and items to compute similarities. Particularly, our model can find similarities of two items by looking for common descriptions (e.g. one actors who plays in two different movies), and by looking for common missing-descriptions (e.g. one actor who does not appear in two movies). We consider that these can be interesting aspects in recommender systems: one user may declare that he does not like sport movies by showing low interest in these movies, however, other users who do not like sport movies simply do not watch/show interests in them.

Chapter 6 focuses entirely on the new users cold-start issue. New users who do not receive pertinent recommendations may abandon the system. In order to cope with this issue, we propose to use active learning techniques into recommender systems. These methods make the new users to interact with the system by presenting a questionnaire that aim to understand their preferences. Questions are related to items, e.g. "do you like this book?". The answers reflect the degree of interest of users in the item, e.g. "yes", "no",

"I have not read it (unknown)". As a consequence, the system can learn from these answers the preferences of users. The goal of active learning is to correctly choose the questions (items) for users. Thus it is necessary to personalize the questionnaires to retrieve the maximum of information possible (i.e. to avoid "unknown" answers). Under this context, we propose an active learning technique that exploits past users' interests and past users' predictions in order to find out the best questions to pose.

Finally, this thesis has been funded under the context of a collaborative project called "FIORA"[3], which aims to build a generic (to be exchangeable to different domains), robust (to be efficient and trustworthy) and scalable (to deal with high quantity of data) recommender system. The applications of this project are oriented to the e-nutrition[4] and e-tourism domains[Cherfi et al., 2016]. Our main goal in this project was to provide a robust collaborative filtering algorithm and to cope with the cold-start issues. Under this context, we have delivered algorithms, development code and technical reports. As long as the FIORA project was in progress and the data from users and items was not available, we decided to use public datasets for the experimentation purposes, e.g. Movielens[5] and IMDb[6]. Thus, in this thesis we only present the contributions made to the field of recommender systems. Particularly, we do not present the FIORA project and we do not show the performances on this project.

[3] http://www.fiora.pro
[4] https://www.mycoachnutrition.com
[5] http://grouplens.org/datasets/movielens/
[6] http://imdb.com

Chapter 2

Recommender Systems: State of the Art

2.1 Motivation

Recommender Systems aim at personalizing the content in webs depending on users' preferences. Their main goal is to predict which items (e.g. movies, books, songs) are of the user's interests and present them first. The challenges that recommender systems have to cope with are very diverse and complex to accomplish them all in one single solution.

[Manouselis and Costopoulou, 2007, Manouselis et al., 2013] have already presented a categorization framework for recommender systems. They have studied around 40 recommender systems and classify them in multiple categories and sub-categories, such as the services provided (e.g. single recommendation, top-K recommendations, find most similar items), the architecture (e.g. centralized, distributed, storage mode, etc) and the recommendation techniques (e.g. user/item classification, data representation, etc). However, this last category is the most extended and analyzed in literature [Burke, 2002, Kantor et al., 2011]. The categorization of recommender systems by the recommendation techniques cares, among other topics, about the data used to perform recommendations, the representation of this data, the algorithms of personalization and the output of recommendations.

A second and more simple categorization of recommended systems was given in [Peis et al., 2008]. The authors propose to divide existing recommendation techniques in three

simple categories: social-based, content-based and economic-based, also called context-based. The social-based methods use similarities between users to compute recommendations. The content-based techniques compute similarities between items regarding their attributes. The context-based methods base their recommendations on concrete particular variables such as the price of items and the budget of users, or the current location of users (e.g. to boost recommendations of currently visited shops).

However, the categorization most widely used simply divides recommendation techniques into Collaborative Filtering techniques (CF), Content-Based (CB) techniques and Hybrid methods [Kantor et al., 2011]. This categorization takes into account the two most common recommendation techniques (collaborative filtering and content-based) and suggests several techniques to combine them [Burke, 2002]. In this chapter we follow this categorization, we explain them and we show their possible variations under an academic and industrial point of view.

This chapter aims to present the related work in the field of recommender systems. Section 2.2 explains the techniques' assumptions and the main background. Section 2.3 presents the approaches made by researches in the academy. On the other hand, Section 2.4 shows some recommender systems in the industry. It is also interesting to compare current existing libraries in order to establish a development reference, which appears in Section 2.5. Finally, we conclude this state of the art in Section 2.6 by giving some discussion about the studied approaches and how our thesis is placed in this context. Other approaches in literature more linked to each specific contribution of this thesis are discussed inside the proper chapters.

2.2 Techniques and assumptions

This section aims to resume the most known recommendation techniques. We aim to lay down the ground to better understand the assumptions of the different techniques and how this affects to the recommendation techniques their-selves. We give special emphasis to collaborative filtering and content-based techniques. Nevertheless, other approaches though further of our scope are also described. Finally, hybrid approaches are also highly

Table 2.1: Recommendation's techniques. Main classification and properties.

Technique	Classification	Advantage	Disadvantage
Collaborative Filtering	Social-based	Domain independence	Sparsity Cold-start
Content-based	Content-based	User independence	Overspecialisation
Context-Aware	Context-based	Items reduction	Complexity
Demographic	Social-based Context-based	Light storage Domain independence	Poor quality
Knowledge-based	Content-Based Context-based	Accurate users' adaptation	Complexity Overspecialisation
Social Networks	Social-based	Particular niches	Complexity
Hybrids	Hybrid	Overcome shortcomings	Complexity More information

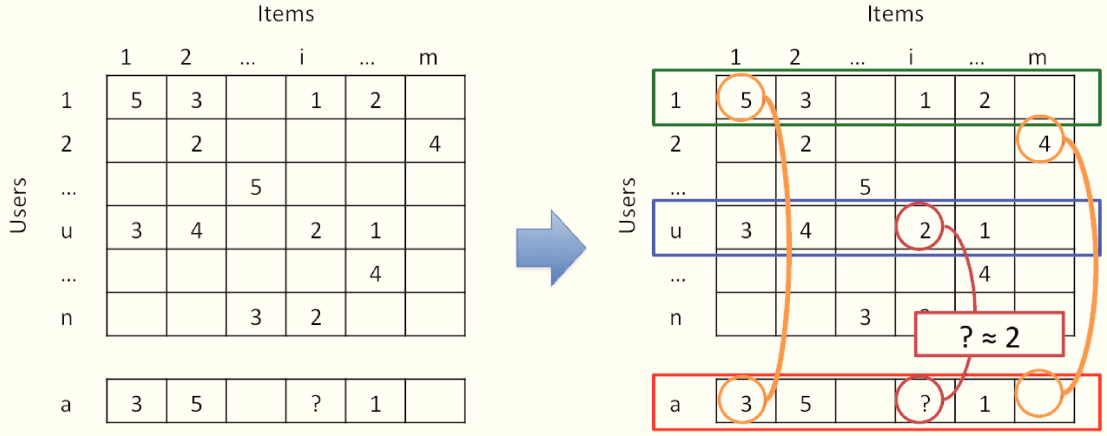
taken into consideration. Table 2.1 gives a brief summary of all techniques.

2.2.1 Collaborative Filtering (CF)

Collaborative filtering is probably the most known and common technique [Shambour and Lu, 2011]. It is based on the assumption that people who share similar preferences in the past would agree also with new preferences in the future. Another interesting point of view is based on the proverb "tell me who your friends are, and I will tell you who you are". As long as these techniques rely on users' correlations, they are also called social-based methods. In brief, these techniques analyze only the users' personal interests (e.g. ratings, clicks) and find out users' sharing similar interests. Hence, it recommends items to users that other users in this group have liked in the past. Figure 2.1 illustrates this process: users rate items (e.g. from 1 to 5) which represent their interest in such item. Thus, we aim to predict tastes for the active user (a) in items and extract the more interesting items to him.

There are two main categories of collaborative filtering techniques: similarity-based (a.k.a. memory-based or neighborhood-based) and model-based (a.k.a. latent factor models) [Breese et al., 1998, Su and Khoshgoftaar, 2009].

Figure 2.1: Collaborative Filtering matching process



On the one hand, similarity-based techniques divide the recommendation process into two steps. First, a *similarity function* is in charge of finding the most similar users to a given one by comparing the items that users have rated and the value of these ratings. Moreover, it represents the closeness of pair of users in a multi-dimensional space of items by using for instance the cosine similarity or the Pearson correlation [Breese et al., 1998]. Second, a *prediction function* uses these similarities, and the past user's ratings to compute ratings' predictions values for the active user by using weighted voting mathematical models, such as the rating average or the biased default voting. Pure user-oriented approaches tend to select items that other similar users liked in the past. A more efficient approach is the item-oriented, in which the recommended items are similar to the ones the user liked in the past [Sarwar et al., 2001]. For further details about voting systems in the recommendation field, readers may refer to [Breese et al., 1998, Schafer et al., 2007a, Su and Khoshgoftaar, 2009].

On the other hand, model-based techniques use probability and algebraic methods to create a user-item model which explains the interests of users and items. These models neatly fits the known data in order to produce rating values for the unknown users-items relations. For instance, Bayesian Networks approaches create a probabilistic model that estimate new recommendations based on ratings' distributions or conditional independence. However, probabilistic models tend to be complex, timely expensive and tedious in case of new incoming data since they need to re-compute the models. Algebraic methods analyze

ratings and create an optimized mathematical model to predict future users and items ratings. They have gained popularity because they are accurate, they avoid dense data storage and they have easy implementations [Koren and Bell, 2011, Konstan and Riedl, 2012]. Other interesting model-based technique use neuronal networks [Kim et al., 2004] and Support Vector Machine (SVM) [Xia et al., 2006].

In general, collaborative filtering have demonstrated their accuracy in exploiting and predicting users' interests. They are considered as "domain independent" techniques [Shambour and Lu, 2011], since no items' description is used, and it represents an advantage in large and multi-domain recommendations (e.g. Amazon e-commerce) because it highly reduces the complexity of the system. However, they suffer from sparsity and cold-start issues.

On the one hand, warm-users have a low number of ratings regarding the whole set of items, what creates a sparse set of data to correlate. On the other hand, new items are not known (rated) yet by users and new users do not have rated items for computing good recommendations. These are the main drawbacks to deal with in collaborative filtering. Some variations focus on how to find and group users of similar interests. For instance, one may take a more restrictive filter into account, such as communities and friend-relationships only [Kantor et al., 2011], and perform users' correlation among this subset of users. In case of new users (from whose preferences are still unknown), to know some of the new users' friends may allow to generate recommendations based on the friend's preferences. As a consequence, it has became popular in social networks as well [Arias et al., 2012].

It is worth to note that model-based techniques overcome similarity-based in accuracy and scalability [Su and Khoshgoftaar, 2009, Koren and Bell, 2011]. In fact, the model construction can be distributed to alleviate time consumption, and recommendations can be stored off-line. On the contrary, similarity-based techniques have to deal with similarity and prediction functions which do not scale well in large datasets.

2.2.2 Content-Based (CB)

Content-based techniques assume that the preference of users do not dramatically change over time, and thus, they recommend similar items to the ones that the active

user liked in the past. These techniques analyze the users' ratings and the items' descriptions. The former allows to know which items the user likes/dislikes. The latter allows to describe the item, and thus, to find similar items.

As it was explained in similarity-based collaborative filtering techniques, content-based techniques also perform a two-step recommendation process. First, a similarity function allows to identify the closeness of items in the database. The description of items can be very large abstract concepts, such as words in text documents, URLs, keywords, or more domain oriented attributes, such as the movie's genre or the author's book; hence this description is composed of very heterogeneous data [Tiroshi et al., 2012]. One very notorious and used technique is the Term Frequency-Inverse Document Frequency (TF-IDF). This technique looks for common words and their word's derivations and count the occurrences of such terms in order to weight how important are words. The second step uses past user's ratings and items' similarities to predict the interest in other items. In this case, predictions have a single user and they are always item-oriented. Thus, voting systems take into account this user only, e.g. the user's average rating.

Content-based filtering is user-independent because there is no users correlation and it bases its recommendations on items similarities only. In addition, the new item cold-start is easy to solve: new item's similarities can be computed and similar items can be recommended to the adequate users. Furthermore, the items' attributes allow to perform easy explicative recommendations: "this item is suggested because you liked this other item". However, they suffer from overspecialisation (or lack of item diversity): these techniques recommend always very similar items, what may bore users who need more diversity in recommendations. In addition, these techniques suffer from new user cold start, since new users have not declared yet enough interests, and thus recommendations are not personalized or diversity is almost in-existent. Moreover, as seen for similarity-based collaborative filtering, these techniques do not scale well in large datasets due to the high volume of computations.

2.2.3 Other recommendation techniques

2.2.3.1 Context-Aware

Researchers have pointed out that the contextual variables from the environment, e.g. GPS location or weather, play an important role in recommendations since they inform about the users' situation [Bonhard and Sasse, 2006]. For instance in a hotel recommendation, it could be interesting to know if the user is going for business or leasure.

They are divided into pre-filtering, post-filtering and context modeling methods. The first approach uses the variables to reduce the number of items to analyze at the input. This helps to reduce the amount of computations and to focus on particularly interesting items. The second method filters the output of the system using the actual context. As a result, it is possible to adapt some details of the recommendations to users, for instance to delete recipes which contains allergens for the user. Finally, context modeling may include the context variables in the computation of recommendations [Kantor et al., 2011], e.g. inside similarity functions, prediction function or complex models. For instance, it is possible to learn not to recommend umbrellas to people visiting Miami in summer.

Pre-filtering and post-filtering techniques are simple to implement because they simply add new layer to the recommender system (at the input or at the output, respectively). On the contrary, context modeling methods may introduce complexity because of the fact of adding new variables to take into account in the recommendation models.

2.2.3.2 Demographic

The demographic filtering techniques are either social-based or context-based techniques. This technique uses demographic personal data to compute a recommendation, such as the age or the nationality [Pazzani, 1999, Vozalis and Margaritis, 2007]. They assume that recommendations should depend on particular niches, for instance, people of the same age may follow similar movie tendencies. These techniques are similar to content-based techniques, although they are user-oriented based on users' personal descriptions. As long as these descriptions are vague and common among the whole community, these techniques lack of precision and are often part of hybrid recommender systems.

2.2.3.3 Knowledge-based

The knowledge-based systems are either content-based, context-based and/or demographical techniques. These techniques use the whole set of knowledge available in order to analyze the interaction between users and items, i.e. rated items, ratings' values and ratings' patterns [Trewin, 2000]. These techniques use to be linked to reasoning systems that apply association rules and other data mining techniques to generate adequate content to be recommended, as well as to explain recommendations.

The two most important techniques are case-based and constraint-based. The main goal of these techniques is to solve a problem for which the solution is a recommendation. The problem has a user description (e.g. explicit queries or implicit needs). Case-based techniques use similarity-based functions to adapt current recommendations (cases) to the user [Bridge et al., 2005, Zhuo et al., 2011]. The case has a static description. The system looks for the closest cases to a given one that best fit the user's needs. On the contrary, constraint-based deal with constraint satisfaction (users or items constraints) [Felfernig and Burke, 2008]. They usually apply association rules over the description of items in order to find out the set of items which answers to the user's problem.

These techniques can be considered complex content-based technique, and they suffer from the same drawbacks as content-based techniques, particularly the overspecialisation issue.

2.2.3.4 Social Network based

These systems are social-based and they are specially linked to collaborative filtering techniques [Kautz et al., 1997, Bernardes et al., 2015]. Collaborative filtering methods assume that users are independent and identically distributed [McPherson et al., 2001]. On the contrary, social network techniques assume that social influences are very important in human choices.

The main difference between collaborative filtering and social network based systems is that the latter take into account more heterogeneous data coming from user social interactions. This has a positive impact on the acceptance of recommended items [Zheng et al.,

2008]. For instance, [Bonhard and Sasse, 2006] stated that carefully controlled familiar profiles and ratings similarities help the system in explanatory recommendations and lead the user in better decision makings.

Indeed, social network techniques focus on the analysis of influence, defined as "the power or capacity of people or things in causing an effect in indirect or intangible ways"[1]. For instance, group of life-friends tend to be friends in social networks (e.g. Twitter or Facebook) and tend to comment on the same subjects and follow the same people. In addition, by using these techniques, it is possible to identify the users who have especial influence over the mass. This has especially raised the attention of the sociology and marketing fields [McPherson et al., 2001], since finding out these people may help in the propagation of publicity and campaigns.

2.2.4 Hybrid methods

The hybrid approaches combine two or more techniques in order to improve the general behavior of the recommender system. It aims to overcome the constraints of one technique with the advantages of the others. For instance, a common hybrid approach is a combination of collaborative filtering and content-based techniques. The former are very accurate and introduce diversity in large number of recommendations, although they suffer from sparsity and cold-start problems. The latter exploit the items' descriptions but they suffer from overspecialisation. As a result, the content-based techniques may use this additional data of items to address cold-start problems and the collaborative filtering introduce some diversity in recommended items.

In [Burke, 2002, Burke, 2007], the author presented 7 different hybridization of techniques, that we will briefly discuss. This categorization has been also well discussed in [Meyer, 2012]. Some examples of hybrid approaches will be given in the next section.

Switching is the most simple hybridization and it consist on selecting one technique to use among set of established recommendations techniques. The choice of the technique depends on predefined rules and the situation, such as the cold-start.

The mixed hybridization proposes to select recommended items from different recom-

[1] http://www.merriam-webster.com/dictionary/influence

mendation techniques and put them together into a recommendation list.

The weighted approach uses "N" recommendation techniques, each of them used under its own assumptions. For instance, to use demographic techniques over users' descriptions and content-based methods over items' descriptions. Then, these techniques separately suggest a set of items to recommend, which can be joint or intersected to create a single recommendation set. The final predicted rating of one item in the set is given by a weighted combination of the predicted ratings from the different techniques.

The cascade hybridization method allows to sequentially enhance the recommendations given by one technique by using another technique. For instance, it is possible to change the order of top-K recommendations given by collaborative filtering by using content-based approaches.

The data's combination (a.k.a. feature's combination) hybridization technique uses the data typically exploited in one recommendation technique into another different recommendation context. For instance, the web logs contain implicit feedback normally used in collaborative filtering, but content based technique may use them together with items' descriptions. The interest of this technique is to discover new usages for data.

The data augmentation (a.k.a. feature's augmentation) technique consists on adding new users or items data (e.g. new ratings), not previously used in other recommendation techniques, before the recommendation process.

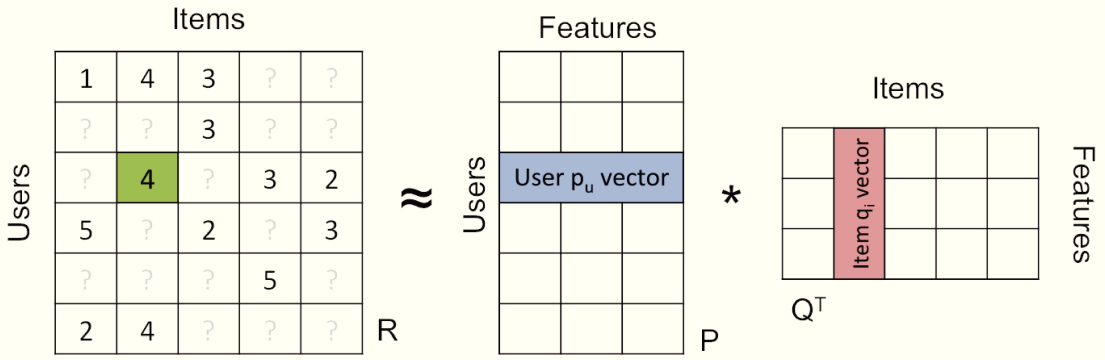
Finally, the meta-level hybridization method provides to a recommendation technique the model generated by a different recommendation technique. For instance, content-based models (ratings and predictions) can be analyzed by collaborative filtering techniques.

2.3 Recommender Systems in the Academy

This section describes the recommendation techniques and the recommendation issues addressed by academic researches. We particularly focus on the scalability and parallelization of techniques, the sparsity and cold-start challenges and the heterogeneity of data used in their approaches.

The first recommender system in 1992 (Tapestry [Goldberg et al., 1992]) was based

Figure 2.2: Matrix factorization of R (users, items, ratings) into P (users, features) and Q (items, features).



on a collaborative filtering techniques. Since then, under our point of view, there is a major technique which has demonstrated a great accuracy in the prediction of users' interest together with a high scalability to distribute the analysis of recommendations among multiple machines. It is called Matrix Factorization (MF) [Koren et al., 2009], a collaborative filtering model-based technique which is related to the Singular Value Decomposition (SVD) approach.

This technique decomposes a matrix into two or several matrices in such a way that their multiplication results in the original matrix (or an approximation of it). These matrices are composed of users-features and items-features, as shown in Figure 2.2. These features are random and represent dimensions in a random-space, what allows to discover latent or hidden relationships between users and items.

The matrix factorization models the interaction between users and items by applying a scalar product of two vectors representing the latent features in a space of dimension f. As a consequence, each item i is associated to one vector q_i, as well as each user u is associated to a vector p_u. The dot product of both vectors represents an estimation of the ratings that the user may give to the item: $r'_{ui} = q_i^T p_u$. Hence, the challenge is to obtain all these vectors q_i and p_u; by solving Equation 2.1, where K is the set of pairs (u, i) in which the ratings of $r_{u,i}$ are available, and λ is a regularization parameter that allows controlling the

learning model.

$$min_{q*,p*'} \sum_{u,i \subset K} (r_{ui} - q_i^T p_u)^2 + \lambda(\| q_i \|^2 + \| p_u \|^2) \tag{2.1}$$

Matrix factorization has been considered a core-stone and it has been used by many authors in literature. For instance, [Takács and Tikk, 2012] propose a ranking-based prediction by minimizing a ranking based objective function instead of the user's rating prediction oriented function. [Hu et al., 2008] has modified matrix factorization to exploit the implicit interest of users in items rather that the users' ratings. They build a binary like-dislike sparsity matrix from an implicit dataset. [Koren and Bell, 2011] suggest similar and enhanced techniques based on matrix factorization. More details of these techniques are given in Chapter 3.

On the other hand, a very famous memory based technique is SlopeOne [Lemire and Maclachlan, 2005]. This item-based collaborative filtering technique suggests to model users' ratings within a slope function. It takes into account users' ratings in items and the ratings received by items to favour the recommendation of popular items. This technique is fast to update and it is resistant to cold start. Other variations consider the high co-rated items [Lemire and Maclachlan, 2005], the bi-polar ("like", "dislike") approaches [Lemire and Maclachlan, 2005], the integration of clustering techniques to improve users' correlations [Mi and Xu, 2011], and apply users' trustiness and items' recommendation usefulness to enhance recommendations [Menezes et al., 2013].

Other collaborative filtering techniques focus on different challenges, such as sparsity. [Wang et al., 2006] suggest a memory-based technique rather than performing users' ratings or items' ratings aggregations to perform users/items similarities, they mix both and they create the rating-based similarity. For a given prediction, both users and items aggregations are used together. [Massa and Avesani, 2004] claims that in large datasets it is difficult to find users who rate the same items with similar ratings. They propose to create a trustiness-graph based on users' ratings and users' connections to increase the number of comparable users.

Some authors focus on these user-user connections and claim that enhancing the seek

of users' neighbors would improve the accuracy of recommendations. [Ortega et al., 2013] propose "Pareto" dominance to perform a pre-filtering process that eliminates less representative users from a K-Nearest Neighbors selection and keep the most similar ones. [Xu et al., 2012] highlight that users with similar tastes in some items may have completely different tastes in other group of items. To face this issue, they propose to find meaningful subgroups by using a multi-clustering techniques over users' ratings and items. This finds accurate cluster of users. [Chatzicharalampous et al., 2015] also realize that users' preferences tend to be distributed among the same group of items. In order to increase the recommendation of other items (coverage) and also have diversity in top-K recommendations, the authors use an user-biased collaborative filtering that favour explorer users in neighborhood memory based techniques.

Other authors agreed with the accuracy of collaborative filtering techniques but focus on more diverse challenges. For instance, [Ben-Shimon, 2013] care about the trade-off between the computation time of the recommendation techniques and the quality of the solution these provide. They propose a memory based collaborative filtering algorithm which can be stopped at anytime. The more the computation time, the better the predictive performance that is achieved. Given sufficient time, the solution becomes optimal. [Herlocker et al., 2000, Hernando et al., 2013] focus on the explanation in recommendations to increase the acceptance of recommended items. They use visualizations and clustering techniques to identify attractiveness and closest related users.

In addition, it is worth to highlight that merging a set of collaborative filtering can overcome simple collaborative filtering approaches [Jahrer et al., 2010], and may co-operate to alleviate each other's particular issues.

Hybrid approaches propose to combine different recommendation technique assumptions, such as content-based and collaborative filtering techniques. [Uchyigit, 2009] and [Peis et al., 2008] show a state of the art in recommender systems that combine semantic web technologies (usually for content-based recommendations) and collaborative filtering. They claim that these technologies may help to interconnect items and users information in other different levels, such as reasoning and recommendation explanations, and they can alleviate cold-start and sparsity. In fact, this may make easier to integrate heterogeneous

data to the system, although it also makes the domain independence and scalability of the system more complex. Many of these techniques suffer from large vector representation of items and/or users which increases the memory consumption.

For instance, [Middleton et al., 2004] create user profiles that represent users' interest in terms of concepts in ontologies. They use collaborative recommendation algorithms to recommend papers seen by similar people and based on their current topics of interest.

[Berkovsky et al., 2007] explore the content-dependency of items proposed by collaborative filtering techniques. They propose an item's partitioning by topic (e.g. the genre of a movie). They apply one collaborative filtering technique per partition by taking into account only the data from users and items in the partition. Differently, [Tso-Sutter et al., 2008, Zhang et al., 2011] enlarge the topics of items. They claim that items' descriptions can be global (i.e. inherit items' attributes, such as the attribute of a movie), or and/or local (i.e. inherit from one particular user for one particular item, such as tags). Their goal is to incorporate tags to different memory based collaborative filtering approaches by creating a "user-tag", "item-tag" and "user-item" matrices. [Mabroukeh and Ezeife, 2011] also exploit the users' tags. They map tags into concepts within one domain ontology. This allows to map items to concepts and to match users to concepts in order to obtain the relevant items for the users.

More approaches regarding hybrid approaches (mainly content-based and collaborative filtering combinations) and the heterogeneity and representation of data are discussed in Chapter 4 and Chapter 5.

As it was discussed in the previous section, the social network techniques may enhance the performance of recommender systems mainly by improving their accuracy, items acceptance and explanation of recommendations [Kautz et al., 1997]. For instance, [Aranda et al., 2007] use a simple social friendship voting system and matrix factorization: the recommendation score for the active user is the sum of the scores of his friends. This enhances the users' similarity research. [Ma et al., 2011] propose to extend matrix factorization to take into account users' social regularizations. They modified the user-features matrix to introduce "social weights" and make that users latent feature vectors and users' friends latent feature vectors to be closer to each other.

Another trending challenge among researches is the cold start situation, specially sensible in collaborative filtering techniques. [Kim and Li, 2004] focus on the new item cold start by using association rule in items' preferences and collaborative filtering probabilistic rating distributions among group of users. Items are partitioned into groups and predictions for users are made by considering the Gaussian distribution of user ratings in each group. [Sobhanam and Mariappan, 2013] use association rules to create users' profile and solve new user cold start and clustering techniques to alleviate the new item cold start. However, current tendencies to cope with cold-start are based on active learning techniques, in which the users can interact with the system and give a few initial ratings to analyze [Boutilier et al., 2002]. The cold start and particularly the active learning techniques are more discussed in Chapter 6.

We conclude this section by presenting some contextual recommender systems approaches. They are interesting because they can adapt recommendations to very particular users' situations[Adomavicius and Tuzhilin, 2011]. For instance, [Sarwat et al., 2014] deduce that there exist users preferences depending on spatial regions (e.g. a school, a neighborhood, etc.). In addition, they discovered that the need to move into different regions looking for special needs is accepted by users. [Braunhofer et al., 2015] propose to identify (or query to users) the contextual factors that could influence users' ratings and recommendations acceptance, e.g., the time or the GPS location. Then, they create a predictive model to predict ratings under various contextual situations.

2.4 Recommender Systems in the Industry

This section presents how recommender systems have been adopted by the industry. We focus on big companies dealing with large amount of users and information. We try to dig into the recommendations that these companies perform. However, this is often difficult due to a lack of public published information, hidden intentions or protected intellectual properties which give competitive edges. In fact recommender systems increase sales (more purchases from catalogues), attract audience (more click on catalogues) and improve users' satisfaction. Thus, the industry cares about the coverage of items and users and the impact of recommendations in the business and the users.

2.4.1 Video Service Providers: Netflix, Youtube and Canal+

Netflix is probably one of the most important actors in the development of recommender systems. This media service provider has organized from 2006 to 2009 the "Netflix Price" competition. The goal was to, given a large movie's dataset, enhance performance of recommendations in terms of the RMSE metric (which computes the square difference between the techniques' predicted ratingsin some items and the ratings the users would have given to these items) by at least 10%. This challenge has contributed by proposing many algorithms (e.g. Matrix Factorization) and many hybrid algorithms, which are combinations of different techniques.

The current recommender engine uses, among others, a linear combination of two very interesting and important techniques: a Singular Value Decomposition for matrix factorization (SVD++) and a Restricted Boltzmann Machines (RBM). The former is explained in [Koren and Bell, 2011]. It is a matrix factorization technique that uses gradient descent techniques to optimize the model research. An extended version was proposed to take into account not only explicit users' feedback (e.g. ratings), but also implicit feedback (e.g. clicks, time spent in items, etc) and other time-oriented users' evolutions. The latter was proposed in [Salakhutdinov et al., 2007]. It is an artificial neural network that learns the distribution probability of ratings among the dataset in order to generate predictions.

Google faces a different problem in video recommendations. For instance, the big ammount of uploaded videos and their poor metadata (e.g. incomplete or irrelevant titles and descriptions). The recommendation system is presented in [Davidson et al., 2010]. It uses contextual informations as well as item-to-item collaborative filtering techniques, i.e. the users who watched this video also watched these others, for long-period recommendations and association rules for short periods of users' navigation (usually 24 hours).

Canal+ aims to offer personalized video content to individuals and groups, i.e. families, in services of "video on demand". They face users interest identification problems as well as video-media players identification (smartphones, tablets, computers and TV's). They developed "Eureka!"[2], a TV programme recommender system of very high content

[2]Recommender Systems Conference at CNAM, Mars 2015. http://www.lesoffrescanal.fr/service-canal-plus/eureka-idee-film

granularity. It adapts constantly programmes depending on interest, wishes and humours. Moreover, it allows to interact with users to enrich its user's preference knowledge.

"Eureka!" captures the devices usages, the different users, the preferences in social networks and the meta-data of their TV programmes. They propose to adapt the content for the family as well by using a cluster approach of TV programmes. This allows to group TV programmes into different categories and to justify recommendations in much easier way. Under a more technical point of view, we consider that this service consists on a set of algorithms to analyze users' interactions with programmes. Thus, they may use a detailed TV programme description and detailed users' experiences interaction to be exploited by content-based and collaborative filtering techniques.

2.4.2 E-commerce: Amazon and eBay

Amazon e-commerce web site offers a very heterogeneous catalogue of products. The recommendation engine is based on collaborative filtering techniques and contextual information. [Linden et al., 2003] popularized the use of item-to-item collaborative filtering recommendations based on log purchases, e.g. customers who bought this item also bought these others items, which are also used nowadays. Rather than matching the user to similar customers, it matches each of the user's purchased and rated items to similar items, then combines those similar items into a recommendation list. Today, a content-based system is also integrated to refine recommendations.

Very recently, Amazon realizes that users may purchase items not for personal usage and that many users keep products in to-buy lists. Thus, it is possible to stop certain purchases from influencing users' recommendations. In addition, Amazon take into account the items in the created lists, as well as the new feature of "wishes" lists.

2.4.3 Social Networks: Linkedin, Twitter and Mendeley

Linkedin[3] aims to connect people and industries in order to help users and companies in job finding and hiring tasks. It offers several recommendation levels. They propose a "job recommendation", in which the user's profile helps to filter and makes appear interesting

[3] www.linkedin.com

jobs. On the other hand, the "talent match" helps human resources to find out interesting candidates. In addition, they offer "news/updates recommendations", "companies you may want to follow", "people you may know", and "similar profiles". These recommendations use collaborative filtering based on users' connections and content based techniques to exploit the users' profiles (such as schools, companies, experiences, skills, etc)[4].

Twitter allows users to post short text messages and pictures. Users can follow other users to be updated about the last posts. The recommendation engine uses users' connections and the short textual information to perform collaborative filtering and content based recommendations [Hannon et al., 2010]. It performs a users' profiling using users' own tweets, users' followees and users' followers. The engine is based on an open source search engine called Apache Lucene[5].

Mendeley[6] is a free reference manager and academic social network. Their goal is to connect users, put researcher in the correct content context and help to find out new publications. The data used comes from Mendeley's libraries, researcher's publications, social network, co-authors network, citations' network, group of researchers and individual profiles. This allows to compute collaborative filtering, content based and social network approach for different use cases, as long as collaborative filtering and content hybrid approaches. For instance, finding similar papers, highlight inter-linked articles, suggesting a group of papers and to find a correct journal to publish an article.

Other social networks, such as Facebook[7], use collaborative filtering to recommend new friends, groups, and other social connections.

2.4.4 Music Service Providers: Last.fm, Pandora and Spotify

Last.fm[8] uses collaborative filtering approaches. It creates a "station" of recommended songs by (1) observing what bands and individual tracks the user has listened to on a regular basis and (2) comparing those against the listening behavior of other users. It

[4] Conference Hadoop World 2011. http://www.slideshare.net/cloudera/2-abhishek-gupta-linkedin-leveraging-hadoop-to-transform-raw-data-into-rich-features-at-linked-in-final

[5] https://lucene.apache.org/core/

[6] Recommender Systems Conference at CNAM, Mars 2014. https://www.mendeley.com

[7] Facebook help system. https://www.facebook.com/help/www/501283333222485

[8] http://www.last.fm

plays tracks that do not appear in the user's library, but are often played by other users with similar interests. Pandora[9], on the contrary, uses content-based techniques to create "stations", which contain songs with similar attributes [Tingle et al., 2010]. It allows positive and negative feedbacks to refine songs in stations. For further details about these recommender systems, readers can read "Pandora and Last.fm: Nature vs. Nurture in Music Recommenders"[10].

Spotify offers different levels of recommendations as well, e.g. songs and other users' songs' lists. They use a content based recommender system[11]. Unlike from other engines, Spotify uses deep learning techniques based on neural networks to generate recommendations [van den Oord et al., 2013].

2.4.5 Other industrial recommender systems

Yelp aims to recommend restaurant to users. They use collaborative filtering to analyze users' ratings and content-based to profile restaurants[12]. They propose a cascade hybrid system that uses a matrix factorization technique and K-Nearest Neighbors, as long as other learning algorithms.

Orange Labs[13] has demonstrated also a big interest in recommender systems to enhance their services. In fact, they propose a similar architecture as Yelp. They propose Reperio, a generic recommender system that aims to help users in decision making, comparison, discovery and exploration [Meyer, 2012]. The recommendation engine uses a K-Nearest Neighbours (KNN) approach based on an item oriented collaborative filtering using a modified Pearson Correlation similarity measure. A combination of KNN and a fast matrix factorization is used to deal with scalability issues. To cope with cold-start, they propose a hybrid technique to deal with content-based recommendations for small users' profile (few ratings).

[9] http://www.pandora.com/restricted

[10] http://blog.stevekrause.org/2006/01/pandora-and-lastfm-nature-vs-nurture-in.html

[11] Blog of Spotify Developer, August 2015. http://benanne.github.io/2014/08/05/spotify-cnns.html

[12] Student Research Study for Yelp.com. http://www.math.uci.edu/icamp/summer/research/student_research/recommender_systems_slides.pdf

[13] F. Meyer working at Orange Labs - Recommender Systems Conference at CNAM, Mars 2014.

2.5 Recommender System's Libraries

Recommender systems have been widely studied in the past few years and have demonstrated their capabilities and accuracy. The academy and industry have both participated in the development of these tools by creating algorithms and proposing new challenges. As a result, there is a great community of researchers and independent users who contribute to make recommender systems more accessible by anyone.

Open source libraries have appeared to create and define a recommendation service architecture. Typically, these libraries do not only focus on recommendations, but they offer data mining, machine learning and data processing paradigms as well. The main goal is to propose libraries to analyze small and very big datasets, by taking into account accuracy and scalability issues.

The most widely known library is Apache Mahout [Owen et al., 2011]. This open source project started in 2010 and it aims to build fast and scalable machine learning applications. On the one hand, the recommendation engine started in the Taste Project[14], which mainly focused on several collaborative filtering approaches in non-distributed environments. This project has evolved inside Apache Mahout by implementing new techniques, such as item based collaborative filtering and matrix factorization approaches. On the other hand, Taste was designed to be run in single machines, although the need to parallelize the data and analysis of recommendation techniques very rapidly arises. Thus, Apache Mahout offers a scalable architecture to perform parallelized analysis on top of Hadoop MapReduce[15]. Currently, it is being adapted to other architectures, like Apache Spark[16], H2O[17] and Flink[18]. For further details about these architectures and their performance in large datasets, readers can refer to [Gopalani and Arora, 2015; Liu, 2015].

There are two recent open source recommender systems which are based on Mahout's libraries: Myrrix[19] and Oryx[20]. Myrrix is a full-stack recommender system, which extends

[14] http://incubator.apache.org/ip-clearance/mahout-taste.html
[15] http://hadoop.apache.org
[16] https://spark.apache.org
[17] Artificial Intelligence. http://www.h2o.ai
[18] Distributed stream flows and batch data processing. https://flink.apache.org
[19] https://github.com/myrrix/myrrix-recommender
[20] Cloudera Oryx Project. GitHub. https://github.com/cloudera/oryx

and improves Mahout to create a scalable recommender system using Hadoop MapReduce. This project is finished and was replaced by Oryx, which has enhanced it and adapted to Apache Spark distribution paradigms.

Another younger (2012) but promising library is MLib[21], which is part of Apache Spark and is easy to deploy in Hadoop architectures as well. It implements a matrix factorization approach based on alternating minimization called Alternating Least Squares (ALS), and this library is growing by implementing more and more algorithms.

GraphLab [Low et al., 2010] is a graph based framework for distributed machine-learning, powered by the open source project PowerGraph [Gonzalez et al., 2012]. It has recently evolved to GraphLab Create in the environment Dato[22]. This library has became proprietary in order to sell services and solutions.

There are some libraries that contribute with some different recommendation features. For instance, CARSKit is the first open-source library to implement context-aware recommender systems [Zheng et al., 2015]. EasyRec[23] [EasyRec, 2013] is a recommendation engine which is wrapped to websites and makes easier to capture, store and analyze the interaction of users and items in the web pages, i.e. implicit interest.

Other libraries are interesting for prototyping algorithms. RecommenderLab [Hahsler, 2011] provides a set of multidisciplinary functions to easily handle users, items and rating in order to develop and test recommender algorithms. It also implements users' and items' similarity based collaborative filtering as well as a matrix factorization technique based on Singular Value Decomposition [Funk, 2006]. LensKit [24] [Ekstrand et al., 2011] libraries allow to build a recommender application and to evaluate new recommendation algorithms as well. The strongest point of this libraries is its community of researchers, since it has collaborated with other GroupLens Research projects.

Other libraries simply implement the current state of the art of recommendation techniques. PREA[25] [Lee et al., 2012] offers several collaborative filtering methods whereas

[21] https://spark.apache.org/mllib/
[22] Machine learning and predictive services in a graph scalable platform. https://dato.com
[23] http://easyrec.org/
[24] http://lenskit.grouplens.org/
[25] http://prea.gatech.edu/

MyMediaLite[26] [Gantner et al., 2011] allows to exploit the item's descriptions as well.

The developer community plays an important role in the development of libraries. This has guaranteed the success of libraries as Apache Mahout. However, the lack of support makes some libraries to stop quickly evolving, such as Carleton [27] Vogoo[28] [Vogoo, 2013], Waffles [Gashler, 2011], Colfi[29] [Brozovsky and Petricek, 2007], MyMedia [Voß et al., 2009], and SVDFeature [30] [Chen et al., 2012].

Finally, there are some libraries which are now inactive or simply outdated, such as Cofi [Lemire, 2003], Duine[31] [Instituut/Novay, 2009], or Crab[32] [Caraciolo, 2012].

Table 2.2 sums up these engines and their properties. An extended and more detailed comparison of some of these frameworks has been very recently published by [Nguyen, 2015]. In addition, [Said and Bellogín, 2014] show the performances of Mahout, Lenskit and MyMediaLite over several collaborative filtering implementations and metrics. They note that same implementations on difference libraries may result different metric values. As a consequence, the choice of which libraries to use become a problem of adaptability to the developer's expertise (e.g. the languages mastered) and the development environment (e.g. to be easily wrapped to other features, such as architectures to distribute the analysis of data). In this thesis, we have specially used Apache Mahout (as long as Myrrix and Oryx) for its simplicity, RecommenderLab for its prototyping features and we briefly analyzed MLib for its integration into Apache Spark architecture.

2.6 Discussions

In this chapter we have explained the assumptions and techniques behind recommender systems. On the one hand, the most popular methods are collaborative filtering and content based. The former relies on the relations among users and users' interest in the past to generate recommendations. It is easily scalable but it does not exploit the items'

[26] http://mymedialite.net
[27] http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/index.html
[28] http://sirius.cs.put.poznan.pl/~inf59829/vogoo/docs/MANUAL.html
[29] http://savannah.nongnu.org/projects/cofi/
[30] http://svdfeature.apexlab.org/wiki/Main_Page
[31] This framework and site are no longer available. http://duineframework.org
[32] http://muricoca.github.io/crab/

Table 2.2: Recommender System Libraries. Table updated on June 10th, 2016.

Library	Recommendation techniques	Language	Last updated
Apache Mahout	Similarity-based CF Model-based CF (Matrix Factorization)	Java	2016
MLib	ALS Matrix Factorization	Scala/Java	2016
GraphLab Create / Dato	Item Similarity-based CF Model-based CF (Matrix Factorization) Popularity based algorithms	Python/C++	2016
RecommenderLab	Similarity-based CF Simple SVD Factorization	R	2016
LensKit	Similarity-based CF Matrix Factorization Slope-One	Java	2016
MyMediaLite	Collaborative filtering	C#/Mono	2016
CARS-Kit	Context aware	Java	2015
PREA	Similarity-based CF Matrix Factorization	Java	2014
EasyRec	Item-based CF	Java	2013
Colfi	Similarity-based CF	Java	2013
Vogoo	Item-based CF	PHP	2013
Waffles	Collaborative filtering	C++	2013
SVDFeature	Model-based CF (Matrix Factorization)	C++	2012
Crab	Component to create RS	Python	2012 - Inactive
MyMedia	Collaborative filtering	C#	2010
	Social Networks based filters		
Duine	User-based CF Content-based	Java	2009 - Inactive
Carleton	Similarity-based Simple SVD Factorization Association rules	Java	2007 - Inactive
Cofi	CF techniques	Java	2003 - Inactive

descriptions to better adapt recommendations. The latter recommends similar items to the ones the current user liked in the past. The heterogeneous data sources increase the complexity of these approaches. In addition, recommendations tend to suffer from overspecialisation (very similar items are always recommended to the users). A third very important category are the hybrid recommendations, which combine two or more techniques to alleviate the drawbacks and enhance the accuracy of recommendations.

Both academy and industry have invested their efforts in the recommendation field. One of the most important contributions is the matrix factorization technique, which appeared during the Netflix price and which has helped researches to overcome with scalability and accuracy issues. Currently, it is considered as the baseline state of the art in collaborative

filtering techniques, and generally, in recommender systems. This technique does not use the items' descriptions and it is sensible to sparsity and cold-start situations. As a consequence, many of the techniques in academy and industry combine it with other approaches.

This thesis assumes matrix factorization as the principle technique in recommender systems and aims to go beyond this technique. We aim to cope with some challenges linked to matrix factorization, collaborative filtering techniques and recommender systems, in general. Our contributions focus on (1) the scalability of matrix factorization to achieve better performance in Hadoop clusters, (2) the implicit interest of users in the items' descriptions and how this can be softly used in collaborative filtering techniques, (3) the items' and users' representations in very large data contexts, and (4) the new user cold start for pure collaborative filtering techniques. Our final goal is that these contributions help recommender systems to better handle data and better understand users in order to enhance the performance of the system and the accuracy of recommendations.

Chapter 3

Analysis of the Parallelization of Matrix Factorization techniques

This chapter is mainly extracted from the conference paper: *"An implementation of a Distributed Stochastic Gradient Descent for Recommender Systems based on Map-Reduce"*; Manuel Pozo and Raja Chiky; IWCIM 2015.

3.1 Motivation

Recommender systems are based on the interaction between users and items. These interactions are given explicitly (i.e. ratings) and implicitly (i.e. tracking navigational behaviour). These sources of information and the large number of users and items imply that recommender systems have to deal with huge data analysis. As a result, recommendations are computationally expensive. For instance, the video media service provider Netflix has around 20 millions customers, 80 thousands movies and 5 billions ratings [Makari et al., 2014]. In addition, this kind of dataset is typically sparse because users often only rate some of these items. Thus, recommenders should be accurate to predict users interests, and scalable to alleviate time processing.

In general, the recommendation techniques focus on accuracy challenges. Matrix Factorization (MF) is a recommendation technique particularly interesting because it has demonstrated great accuracy in recommendations and a high scalability to suit very large users/items datasets [Koren, 2009, Koren et al., 2009]. This technique has become the main reference in the state of the art. The two most known matrix factorization based

techniques for recommender systems are Alternating Least Squares (ALS) and Distributed Stochastic Gradient Descent (DSDG). We particularly focus on the latter one because it has demonstrated better accuracy [Makari et al., 2014]. However, the distribution of the computation among multiple machines is complex to achieve and the current technique to parallelize the process has a negative impact in the accuracy of the system.

In this chapter we analyze the scalability of matrix factorization. Our contribution proposes a modification of the current distribution of stochastic gradient descent techniques for a Hadoop/MapReduce cluster. Our goal is to render more flexible the distribution of these techniques in order to better exploit the capabilities of the cluster and to achieve a high accuracy in recommendations. At the best of our knowledge, this paradigm has not been openly detailed under this context. Thus, we detail the implementation of our approach. The experimentation has been performed using the publicly available MovieLens dataset. The results indicate the good performance of the system compared to current state of the art.

This chapter focuses on matrix factorization techniques from the point of view of scalability and accuracy. Section 3.2 starts by presenting a brief state of the art. The necessary background to understand the troubleshooting in the distribution of stochastic gradient descent is given in Section 3.3. The contribution and implementation details of our approach are in Section 3.4. The experimentations and comparisons are in Section 3.5. Finally, we conclude and discuss this chapter in Section 3.6.

3.2 Related Work

This state of the art focuses on scalable recommendation techniques. Particularly, collaborative filtering approaches based on matrix factorization technique have demonstrated high scalability and high accuracy predicting recommendations [Koren, 2009]. In brief, this technique relies on collaborative filtering assumptions: users who agreed in past tend to agree in future. Hence, it groups people of similar tastes, and it recommends past liked items from people with the same preferences. On the contrary, it does not exploit the items' and/or users' attributes to enhance recommendations.

The matrix factorization technique decomposes a matrix R into two random matrices, P and Q, in such a way that the multiplication of both matrices gives approximately the original one [Koren et al., 2009]. Typically, $r_{u,i}$ denotes known ratings in R, where u stands for a user (row) and i for an item (column). Thus, ratings are obtained by the multiplication of a vector p_u from matrix P and a vector q_i from matrix Q: $r_{u,i} = p_u \cdot q_i^t$. By using this technique, known ratings are approximated or unknown ratings are predicted: $\hat{r}_{u,i} = p_u \cdot q_i^t$.

The quality of the model is given by the closeness between approximated ratings $\hat{r}_{u,i}$ $r_{u,i}$ and real observed ratings $r_{u,i}$. The goal is to find the matrices P and Q that best approximate known ratings in R: the technique looks for the best P and Q that minimize the quadratic error of the difference between real and approximated ratings. The baseline model is then defined as: $min \sum_{r_{u,i} \in R} (r_{u,i} - \hat{r}_{u,i})^2$.

As a consequence, the matrix factorization paradigm becomes an optimization problem to solve. The two most known optimization techniques that may find out accurate predictions are based on alternating minimization and gradient descent [Koren and Bell, 2011].

3.2.1 Alternating minimization for the matrix factorization

The alternating minimization technique has simple algebraic resolution. It was popularized by the Alternating Least Square (ALS) technique, studied in [Schafer et al., 2007b, Zhou et al., 2008, Pilászy et al., 2010, Takács and Tikk, 2012, Jain et al., 2013].

This technique decomposes the problem into two simple optimization problems represented in P and Q. The main idea is that knowing the ratings in R and supposing P or Q fixed, the non fixed matrix can be guessed. By alternating the fixed matrix in order to guess the other one yields in an approximated result for R. Thus, one alternating minimization iteration is completed when P and Q have been guessed/fixed one time. Highlight that this alternating minimization exploits R two times in one iteration: to guess P and to guess Q. New iterations repeat all this process until a maximum number of iterations or a convergence threshold has been achieved.

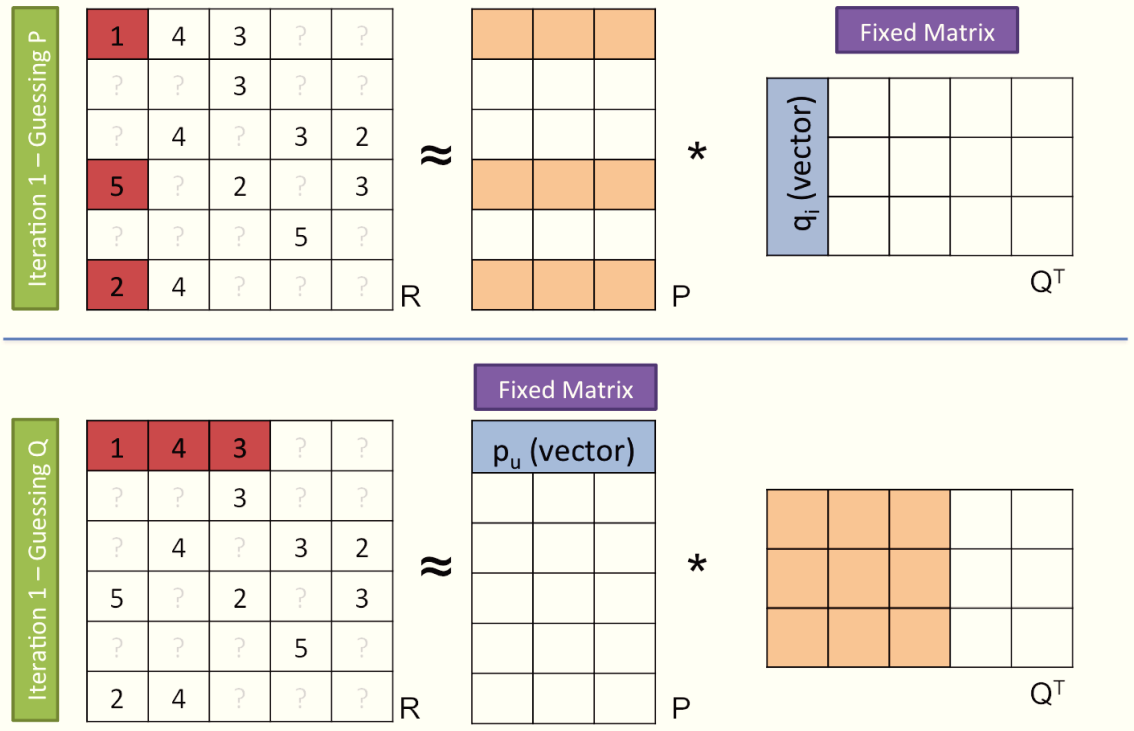


Figure 3.1: Alternating Least Squares technique. Methodology and scalability.

The scalability of this technique relies on the independence in the optimization problem. Let's consider that one iteration is a 'job' to execute, which contains two known data sources (i.e. the fixed matrices) and one set of variables to solve (i.e. the guessed matrix). For instance, let's have R and P fixed in order to guess Q. As long as (1) items' are independent (colmuns in Q are independent), (2) users' latent features are fixed, and (3) there is no common learning parameter dependency to update, it is possible to decompose the job into more simple tasks, a.k.a. "works". Thus, one work aims to guess one single column of Q using related ratings and users' ratings for this item's column. As a consequence, works can be easily computed in parallel. This shows a highly level of distribution and schedule of works, jobs, iterations and the algorithm in general. Figure 3.1 represents this technique and its scalability.

3.2.2 Gradient descent minimization for matrix factorization

The gradient descent optimization technique was popularized as Gradient Descent (GD) and Stochastic Gradient Descent (SGD) for recommender systems in [Funk, 2006, Takács et al., 2007, Koren, 2008, Koren, 2010, Koren and Bell, 2011]. In other references it is also known as Singular Value Decomposition for recommender systems.

This technique is typically more complex. It decomposes R into two matrices P and Q and add dedicated users/items learning-parameters to study the ratings patterns. This technique iterates over each rating $r_{u,i}$ in R, one by one, looking for a global minimum. After each rating, the u user's parameters and the i item's parameters are updated by taking the negative gradient of the function into account. These little steps (single rating and consequent update) improve the accuracy of the system towards the global minimum. One algorithm's iteration is completed when all ratings have been analyzed. One may notice that huge number of ratings may cause slow time performances. To solve this, the stochastic gradient descent suggests a faster convergence optimization: it iterates over a batch of ratings before updating the parameters in matrices. This affects to the quality of the convergence[1], although it has minor effects in very large ratings datasets.

The complexity of the scalability of these techniques relies on the ratings iterations and updating steps. In fact, the iteration over ratings and the followed updates are inter-dependent because they use and cross users and items parameters. If two ratings sharing similar users or items are computed in parallel, the users/items update step represents a problem. This paradigm is detailed in [Gemulla et al., 2011, Makari et al., 2014].

The main idea behind the parallelization of this algorithm is the division of the R matrix into *blocks*, *stratums* and *iterations*. Figure 3.2 represents an example of the presented paradigm, which is described below.

A *block* is a batch of ratings that should be computed. It is computed by only one processor, and hence, the update of parameters are available within the same processor and there is no dependency problems inside[2]. It is possible to note that there are blocks

[1] see `http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/` for further details.

[2] We consider that processors do not directly share memory.

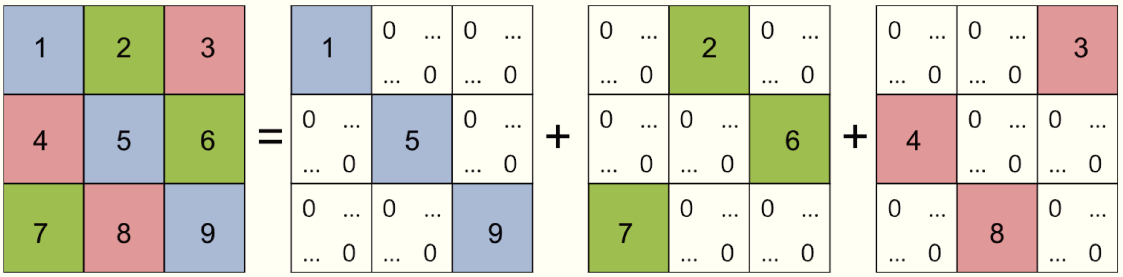


Figure 3.2: Stratification and Block decomposition. Blocks 1,5,9 do not share rows neither columns, thus, they form a stratum to run in parallel. Once computed, another stratum (2,6,7 or 3,4,8) can run. Single iteration.

which do not share rows (users) neither columns (items). These independent blocks can be computed in parallel.

A *stratum* is defined as a set of independent blocks. Then, inside a stratum there are no dependency problems. The blocks in a stratum can be computed in parallel in different processors, machines or clusters.

Finally, one *iteration* is the execution of all stratums, where stratums can be computed in sequence or arbitrary sequence [Makari et al., 2014]. The algorithm finishes after all iterations have been processed. As a result, this technique is stratified and safely parallelizable.

3.2.3 Comparisons

Gradient descent techniques are more accurate than alternating minimization techniques, as denoted in [Koren and Bell, 2011, Makari et al., 2014]. However, they are more complex to distribute as well. As a consequence, there is a tendency to use alternating minimization due to their good accuracy and easy scalability, as deduced from [Gemulla et al., 2011, Jain et al., 2013]. Moreover, this technique has been tested in GPU parallelization to improve its time-consumption [Kampffmeyer, 2015] .

The recent distributed approach of gradient descent in [Makari et al., 2014] demonstrates that it is possible to overcome alternating minimization in both accuracy and scalability. This technique is not well detailed for Hadoop/MapReduce clusters. In addition, the paradigm approach may yield to a loss of accuracy due to a loss of ratings in the block

creation process.

Our contribution aims to tackle this problem by slightly changing the distribution paradigm and to offer an implementation of a distributed stochastic gradient descent.

3.3 Background

This section goes deeper into the stochastic gradient descent technique in order to better understand the troubleshooting in its distribution for scalable recommender systems.

Gradient Descent (GD) for recommender systems is based on the matrix factorization technique. It aims to explain the ratings in a matrix R by using two matrices P and Q. The baseline predictor is defined as $\hat{r}_{u,i} = \mu + b_u + b_i + p_u \cdot q_i^t$. Where μ is the average of the ratings in the matrix R. b_u, b_i are biases of the user u and the item i respectively and represent a deviation from the average rating value. Finally, p_u, q_i are the latent space vectors from P and Q matrices. The goal is to minimize the λ-regularized squared error of:

$$min_{b_u,bi,p_u,q_i} \sum_{r_{u,i} \in R} (r_{u,i} - \hat{r}_{u,i})^2 + \lambda \cdot (b_u^2 + b_i^2 + \|p_u\|^2 + \|q_i\|^2)$$

The gradient descent optimization analyzes all the ratings one by one by computing the error in the prediction, $e_{u,i} = r_{u,i} - \hat{r}_{u,i}$. It updates the current parameters' status after each rating by taking the negative gradient. This process is controlled by a learning-rate parameters called γ. The update phase is given by:

- $b_u := b_u + \gamma \cdot (e_{u,i} - \lambda \cdot b_u)$
- $b_i := b_i + \gamma \cdot (e_{u,i} - \lambda \cdot b_i)$
- $p_u := p_u + \gamma \cdot (e_{u,i} \cdot p_u - \lambda \cdot q_i)$
- $q_i := q_i + \gamma \cdot (e_{u,i} \cdot q_i - \lambda \cdot p_u)$

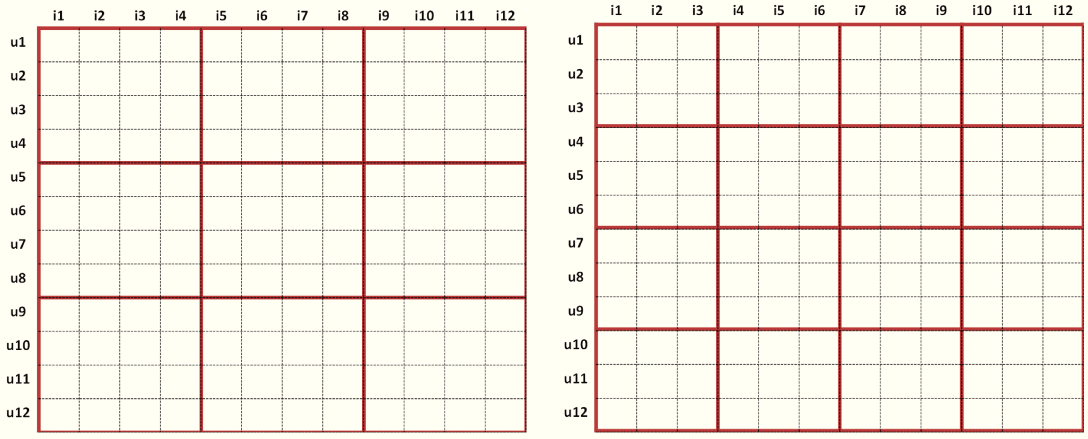
The stochastic version proposes to iterate over a batch of ratings before updating the parameters, what allows a faster convergence in very large datasets. However, either simple

iteration or batch iterations, one can highlight the interdependency of the new updated values and previous values [Makari et al., 2014]. For instance, a new value for b_u depends on the last computed value of b_u for this user u. In a single machine, new updated values after one iteration are ready for a new iteration. On the contrary, non sharing memory systems may have difficulties to scale the algorithm and it may lead in miss-updated values or lack of synchronization.

[Gemulla et al., 2011, Makari et al., 2014] have demonstrated that it is possible to avoid this dependency by looking for independent ratings, i.e. ratings $r_{u,i}$ that do not share rows (users) and/or columns (items). As a consequence, they have no common parameters; hence, there is no dependency; and thus, they can be computed in parallel. As explained above, the distribution of this technique follows a stratification paradigm. One stratum is composed of blocks. One block is composed of a batch of ratings. Blocks in the same stratum do not share rows (users) and/or columns (items), i.e. one user/item belongs only to one block in a given stratum, and thus one user/item is contained in only one block per stratum. On the one hand, one block is computed in one node, i.e. ratings are in the same machine, and it is possible to safely update parameters. On the other hand, blocks in one stratum do not share users/items and thus can be computed in separated machines, and the updates for future stratum's analysis are possible and the interdependence is avoided.

We highlight that the performance of the distribution depends on the number of independent blocks and the number of nodes to analyze the block. We assume that one node analyzes only one block at a time, and that one node is uni-threaded (not divisible). For instance, one node can be one cluster in a multi-machine environment, or one thread in a multi-threaded single machine. Hence, having b nodes, the goal is to find b independent blocks in the matrix to maximize the efficiency of the distribution. For instance, let's represent a matrix R of dimensions $n_u = 12$ users (rows) and $n_i = 12$ items (columns). Figures 3.3(a) and 3.3(b) show the block decomposition when 3 and 4 nodes ($b = 3$ and $b = 4$) are available. One can highlight that the size of block has been changed and adapted due to the number of nodes.

However, (1) what would happen if there are 5 nodes in the cluster?, and (2) what would happen if R is not a square matrix, $n_u \neq n_i$? In fact, the key point for the block



(a) 3 blocks per stratum. (b) 4 blocks per stratum.

Figure 3.3: Diving the matrix R into independent blocks per stratum

decomposition is the divisibility of the dimensions in integer parts: the dimensions of this matrix might be divisible by b in order to properly create blocks, i.e. $n_u\%b = 0$ and $n_i\%b = 0$. Highlight, that **matrices do not necessary need to be square for a block decomposition**, but once the size of blocks is defined, **the block itself becomes a square sub-matrix**. This integer divisibility constraint is a conflict for a block decomposition and to adapt the decomposition of block to the cluster. In this work we aim to tackle this issue, what we call "divisibility condition" or "decomposing condition".

3.4 Flexible Distributed Stochastic Gradient Descent

Our work focuses on coping with the divisibility condition in the block decomposition of the distributed stochastic gradient descent. [Gemulla et al., 2011], [Makari et al., 2014] have suggested the deletion of rows/column (i.e. users/items) to achieve it. Hence, when $n_u\%b \neq 0$ and/or $n_i\%b \neq 0$, the authors delete users and/or items in order to achieve $n_u\%b = 0$ and/or $n_i\%b = 0$. However, this may yield in a loss of accuracy: if a significant number of data is deleted the system may loose important information, i.e. ratings.

Our contribution relies on the flexibility in the condition accomplishment. On the one hand, deleting dimensions loses data. On the other hand, adding dimensions may slightly increase the time-consumption of the algorithm since more clusters communications

might be performed. However, this does not yield to a loss of data, since inserting empty dimension does not add any noisy information. Based on these ideas, we have developed a flexible distributed stochastic gradient descent.

In this section we first formalize the flexibility aspect of the distribution, and second we focus on development and implementation details of the approach.

3.4.1 Flexibility and adaptation of blocks to the clusters

Let's n_r be the number of rows (users) and n_c the number of columns (items). b is the number of nodes, machines or clusters, and thus, the number of possible independent blocks. Our goal is to define new n'_r and n'_c in such a way that $n'_r \% b = 0$ and $n'_c \% b = 0$. In fact, by doing this we set the number of blocks in which the matrix R will be decomposed and the size of these blocks.

The under dimensionality setting deletes rows and/or columns, and thus users/items; hence, ratings and information. Highlight that gradient descent usually performs a random shuffling in the matrix R, and thus which users/items are deleted is not controlled. This mode resizes the matrix by deleting dimensions, and thus resized dimension are equal or minor to the real one to accomplish the decomposing condition. This is also used in [Makari et al., 2014]. As a result, the number of rows/columns to use in the resized matrix is given by $n'_r = n_r - n_r \% b$ and $n'_c = n_c - n_c \% b$.

The upper dimensionality setting adds new rows and/or columns, and thus users/tems; however, these new dimensions do not contain ratings. This may possibly create an unbalanced block in every stratum in which the number of ratings is lower. As a result, it may cause extra time consumption due to the fact that the block needs to be sent to the node anyway. However, this time is normally insignificant compared to the time-consumption of the algorithm and it does not carry out any loss of ratings and hence any loss of accuracy. The number of rows/columns to keep in the matrix is given by $n'_r = n_r + (b - n_r \% b)$ and $n'_c = n_c + (b - n_c \% b)$.

The flexible dimensionality setting is a hybrid between both approaches above. It deletes/adds users and/or columns depending on the number of nodes b. It looks for the

Algorithm 1 Looking for the closest value which return zero remainder.

```
1: function ClosestZeroRemainder(dimension, rule)  ▷ rule is the number of nodes available.
2:     underResidual = dimension % rule;
3:     upperResidual = rule - dimension % rule;
4:     if underResidual < upperResidual then  ▷ Picking up the closest for zero remainder.
5:         closest = dimension - underResidual;
6:     else
7:         closest = dimension + upperResidual;
8:     end if
9: end function
```

closest dimension (above or below) to the real one. Whether the algorithm deletes or adds rows/columns depends on the remainder $n'_r \% b = 0$ and $n'_c \% b = 0$. Algorithm 1 shows the process to find the closest dimension either below (under) or above (upper) the original dimension. The "rule" is the number nodes b which is the divisor to divide dimensions towards a zero remainder.

3.4.2 DSGD Hadoop/MapReduce implementation

In this section, we present the technical details of our implementation in MapReduce. Three parameters have to be set: row original matrix dimension n_r, column original matrix dimension n_c, and the number of nodes b (non divisible threads or clusters). We focus on three parts of our implementation: (1) the block decomposition of the R matrix, (2) the stratum assignment of decomposed blocks, and (3) the execution of these stratums and the end of the algorithm.

3.4.2.1 Matrix Block Decomposition

The first goal is to decompose the R matrix into blocks, in such a way that independent blocks can be found and grouped into stratums. Our approach allows three decomposition modes, which have been explained before. This step takes the chosen mode and applies it. As a consequence, this step adds/removes rows and/or columns in order to accomplish the decomposition condition and make the rating matrix decomposable in blocks.

Furthermore, solving the decomposing condition makes possible to determine the size

Algorithm 2 Decomposition of the matrix R into blocks.

```
1: function BlockDecomposition(R, numberOfNodes, mode)        ▷ mode can be:
   underDimension, upperDimension or flexDimension.
2:     rowDimension = decCondition(R.rowDim, numberOfNodes, mode);
3:     columnDimension = decCondition(R.columnDim, numberOfNodes, mode);
4:     R' = makeDecomposable(rowDimension, columnDimension, R);
5:     rowStepSize = (rowDimension/numberOfNodes);
6:     columnStepSize = (rowDimension/numberOfNodes);
7:     for row = 0; row ≤ rowDimension; row++ do
8:         for column = 0; row ≤ columnDimension; column++ do
9:             createNewBlock(row, rowStep, column, columnStep, R');
10:        end for
11:    end for
12: end function
```

of blocks given R and the number of nodes. This is useful to go all over the matrix R extracting the corresponding blocks. The row step size of a block is given by $rowStep = \frac{n'_r}{b}$ and the column step size is given by $columnStep = \frac{n'_c}{b}$. Algorithm 2 describes the complete block decomposition process. First we find the appropriate number of rows and columns (regarding the underDimension, upperDimension or flexDimension) which fit with the number of nodes. Second, we shrink or enlarge the original matrix R according to these dimensions. Finally, the new dimensions and the number of nodes allow to compute a step size to follow to build blocks in the matrix.

3.4.2.2 Stratum Assignment

The second step is to find out the independent blocks and to assign them to the same stratum. Note that, the number of the stratums is actually the number of nodes b, and thus, up to b blocks are in the same stratum and can be executed in parallel.

We use a straightforward strategy: blocks found are numbered one after the other, which are used as blocks identification and to find out blocks inter-dependencies. In fact, the decomposition of the matrix has been done per column step and per row step, in this precise order. This makes identificators to increase first by column and second by row. Figure 3.4 exemplifies this explanation.

The stratum assignment takes benefit of this numbering by implementing a priority list.

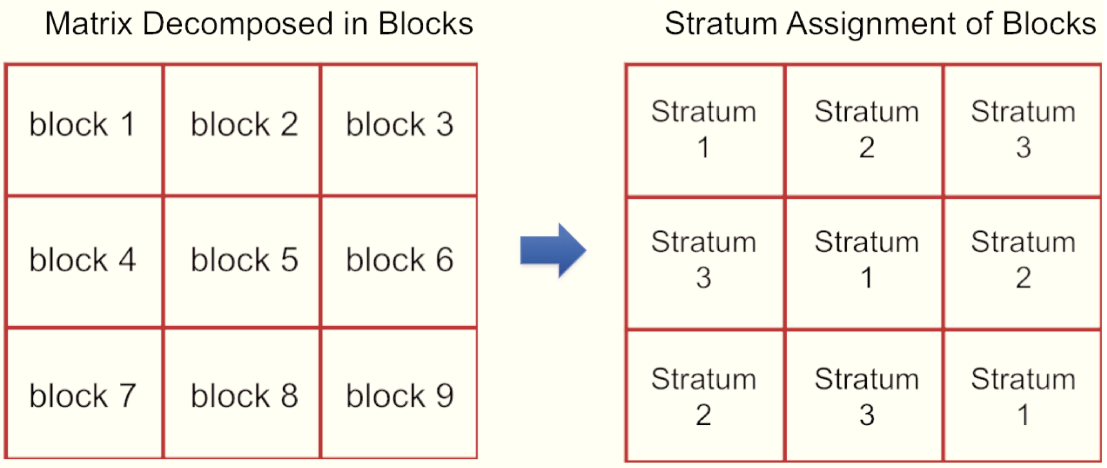


Figure 3.4: Block Stratum Assignment technique.

Algorithm 3 Assignment of independent blocks to stratums. Assignment is made first by columns and second by rows.

```
1: function StratumAssignment(blocks, numberOfNodes)   ▷ Note that numberOfNodes is equal to the number of independent blocks, and stratums.
2:     priority                                          ▷ vector or list of size numberOfNodes
3:     for stratumID = 1; stratumID ≤ numberOfNodes; stratumID++ do
4:         priority.addStratum(stratumID)
5:     end for
6:     for blockID = 1; blockID ≤ blocks.numOfBlocks; blockID++ do
7:         stratum = priority.getFirstElement()
8:         stratum.addBlock(blockID)
9:         priority.rotate(1)   ▷ rotate the order in the priority vector/list by one, pushing by one position only. (either to the left of the right)
10:    end for
11: end function
```

This list contains the stratums in such an order that the first stratum in the list is always the one to accept a new block. After each block insertion, this list rotates by one position assuring that all stratums are filled sequentially. Algorithm 3 describes this process.

3.4.2.3 Stratums' execution, number of iterations and end of algorithm

The third step is to execute the stratums. The blocks inside the same stratum are independent. Although blocks of different stratums are dependent. Hence, stratums are dependent among them, and thus they have to be run in sequence. However, the order of

Algorithm 4 Execution of iterations and stratums. P and Q are the users/items features matrices respectively.

```
1: function RunIterations(P, Q, maxIterations, numberOfStratums)   ▷ Note that
   numberOfStratums is equal to the number of nodes.
2:     for iteration = 1; iteration ≤ numIterations; iteration++ do
3:         for stratum = 1; stratum ≤ numberOfStratums; stratum++ do
4:             doSGDJob(stratum, P, Q);
5:             updateParameters(P, Q);
6:         end for
7:     end for
8: end function
```

this sequence can be random.

Running one stratum means to compute the stochastic gradient descent among the ratings in blocks and then updates parameters in P and Q at the end of their execution. This guarantees the integrity and independence of parameters. Algorithm 4 describes this pseudo-code.

The *doSGDJob* is a MapReduce job that analyzes the ratings in the blocks within a stratum to compute the gradient descent optimization. One block is sent to a different cluster and thus one block is defined by one single map. However, all stratum's blocks are computed in parallel. Hence, there are b clusters executing b different blocks, which are b map tasks.

The $Map - function$ uses a simple key-value pair as input. The key is a pair of values itself, which represent the entries of the R matrix (row/column identification or user/item identification). The value is the rating encoded in a double variable. This function iterates among ratings and internally updates the users/items parameters, however it does not notify this update. Once all ratings have been analyzed, a clean-map process is computed to prepare the final output of the function. The output is composed of a key-value pairs as well. The key is a pair of values representing the user or item identification. The value is a vector which contains the updated user's/item's parameters. This paradigm makes unnecessary to use $Combiners$ or $Reduce - functions$. Finally, the parameters are joint to the original P and Q matrices and updated locally for next stratums.

The end of the algorithm is given by number of iterations to compute. One iteration

is finished when all stratums have been computed. As a consequence, this makes stratums to be computed several times and to enhance the accuracy of recommendations after each iteration.

3.5 Experimentation

The experimentation phase uses the publicly available MovieLens 10M dataset [3]. It contains 69878 users, 10667 movies and 10 million ratings. We focus on collaborative filtering techniques only, and thus there is no domain data or external information to aggregate. By using this dataset, we create a matrix of rows (users) and columns (items) that contains the available ratings. The evaluation and comparisons are carried out in terms of accuracy and scalability as follows.

On the one hand, we split the dataset into 90% training set and 10% test set. This split is done randomly by ratings, and thus one user/item can be both in training and test set, although the rating of one user in one item belongs to one of the sets only. In order to evaluate the accuracy of the recommendations, we use Root Mean Square Error (RMSE). This metric computes the square error between the difference of predicted values and real observed values. As a consequence, we use the training set to train and create a recommendation model, and the test set to evaluate the performance of the model. As long as the split of the dataset is random and affects to the creation of predictive models, we split, run and evaluate algorithms 10 times. Then, we take the average value of the RMSE to compare the techniques.

On the other hand, the time-consuming and scalability parameters depend on machines and clusters. We vary the number of nodes in the cluster and see how the recommender systems perform. We use $b = 1, 2, 5, 7, 15$, which affect the number of machines to use as well as the number of blocks in which we decompose the matrix of ratings. In this analysis we run the experimentation in single node cluster which uses a MacOS 4Go RAM with 2 cores (2.53GHz). This makes mappers in map-reduce to be computed sequentially, thus time-consumption is higher than in a multi-cluster environment.

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

We compare the performance of the three stochastic gradient descent explained in this chapter: under dimensionality (underDSGD), upper dimensionality (upperDSGD), and flexible dimensionality (flexDSGD). Highlight that underDSGD is the technique in [Makari et al., 2014]. All these techniques use similar training parameters obtained by cross-validation: overfitting is $\lambda = 0.025$ and learning rate is $\gamma = 0.0075$. In addition, the matrix factorization is allowed to run 30 iterations and to create a latent space model of 30 features. Moreover, we offer a comparison to the alternating minimization technique ALS in Apache Mahout[4], implementation of [Schafer et al., 2007b, Zhou et al., 2008]. This probes the superiority of distributed stochastic gradient descent techniques in terms of accuracy and scalability. The technique's settings uses the overfitting $\lambda = 0.025$, obtained by cross-validation. In addition, in this case the technique is also allowed to run 30 iterations and to use 30 latent features.

It is expected that upperDSGD obtains slightly better results in terms of RMSE, since it does not delete data. Yet, underDSGD might get slightly better time performance. FlexDSGD will show intermediary results in both comparisons. Table 3.1 shows the performance in terms of RMSE and the time achieved in computing one iteration in a *single node cluster* are presented. Moreover, it shows the number of rows/columns (users/items) that has been added or removed (denoted by "+" or "-", respectively) in order to achieve the divisibility "condition": $n_u\%b = 0$ and $n_i\%b = 0$.

As it was expected, the stochastic gradient descent techniques perform better than alternating minimization techniques showing better accuracy and faster computations. Highlight that the time show in Table 3.1 when $b > 1$ correspond to the time of executing b blocks in sequence as long as this experimentation was perform in a single machine. In real multi-clusters environment this time should be exponentially reduced while increasing the number of nodes, until finding an asymptote. In addition, the upperDSGD and flexDSGD approaches slightly overcome the underDSGD in terms of accuracy. Indeed, these two modes deal with more ratings, what allows achieving (tiny 1%) better results. This fact explains as well the little extra time taken in computation. Highlight that less or none rows/columns are deleted. For instance, focus in the case of 15 number of units, which has

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

Table 3.1: Experimentation results.

Technique	RMSE	Time (min)	Number of nodes (b)	Rows	Columns
ALS	0.79603	3.019	1	=	=
Under DSGD	0.77571	1.303	1	=	=
Flex DSGD	0.77571	1.303	1	=	=
Upper DSGD	0.77571	1.303	1	=	=
Under DSGD	0.77611	1.850	2	=	-1
Flex DSGD	**0.77555**	1.852	2	=	+1
Upper DSGD	0.77559	1.885	2	=	+1
Under DSGD	0.77626	5.518	5	-3	-2
Flex DSGD	0.77617	5.524	5	+2	-2
Upper DSGD	**0.77597**	5.525	5	+2	+3
Under DSGD	0.77586	8.978	7	-4	-6
Flex DSGD	**0.77548**	8.985	7	+3	+1
Upper DSGD	0.77565	9.024	7	+3	+1
Under DSGD	0.77593	28.506	15	-8	-2
Flex DSGD	0.77596	28.526	15	+7	-2
Upper DSGD	**0.77555**	28.536	15	+7	+13

15 independent blocks per stratum. In order to achieve this decomposition, underDSGD has deleted 8 rows (users) and 2 columns (items), yielding in a loss of data. In addition, the deletion is not controlled, and thus one may delete very informativeness rows and columns.

3.6 Discussion

The current most widely used recommendation technique is matrix factorization because it is scalable and accurate. This technique decomposes the users' ratings in items into two matrices, in such a way that the multiplication of both matrices result (an approximation of) the original one. Finding these two matrices becomes an optimization problem which can be used using alternating least squares and gradient descent techniques.

In addition, the large quantity of users, items and ratings highlight the need of a scalable and distributed technique to analyse and process this data. In this chapter we have analyzed the scalability of alternating least squares and gradient descent techniques for recommender systems. We highlighted that the latter was more complex to distribute although it leaded to slightly more accurate recommendations.

The current distribution paradigm for gradient descent does not properly parallelize the algorithm among an existing Hadoop cluster of machines because it may lead to a loss of users, items and ratings. This causes a light loss in accuracy. We have developed a Distributed Stochastic Gradient Descent (DSGD) algorithm based on this technique and this paradigm that solves this issue. This technique also overcomes alternating minimization optimizations of the matrix factorization problem. The experimentation phase uses the public MovieLens dataset. The evaluations show the good performance of the approach in terms of accuracy and scalability.

One important fact for future work is the capability of the SGD to be extended. One may incorporate more heterogeneous data (e.g. implicit feedbacks and timestamps) in order to improve its accuracy. This has proved to increase the accuracy of recommendations in several techniques. In addition, this is a complex challenge in collaborative filtering recommendation techniques. The next chapter (Chapter 4) suggests a solution to incorporate the users' interests in the items' attributes into collaborative filtering approaches, although reducing the impact of the domain dependency in recommendation processes.

Chapter 4

The implicit interest of users in the attributes of items

"My mom used to tell me stories at night, read books to me - and I read 'em over and over and over again. And you know what I learned from that? I went back and looked at everything - Why do I like reading the same stories over and over and over again? What, was I some kind of nincompoop? No - the narrative gave me connection with my mom."

- Peter Guber (Chairman and CEO of Mandalay Entertainment.)

This chapter is mainly extracted from the journal paper: *"Enhancing Recommender Systems by Exploiting the implicit interest of users in items"*; Manuel Pozo, Raja Chiky and Elisabeth Métais; TCCI 2015.

4.1 Motivation

Recommender systems exploit the interest of users in items to recommend other items, unknown by the users, which could be of their interest as well. There are many approaches to retrieve and deduce the users' interest, a.k.a. feedback, although they can be summarized into explicit and implicit. Both of them are item-oriented, i.e. the interest of the user in the item.

The current recommender systems use these feedbacks to generate recommendations. However, these feedbacks show only the interest of the user in the item (e.g. movie, song, book) and not in the item's attributes (e.g. actor, composer, writer). In addition to this, users are not willing to rate many items, and thus asking for rating items' features is

not suitable [Elahi et al., 2014]. Similarly, the implicit feedback captured from users are typically item-oriented as well [Oard et al., 1998]. As a consequence, we highlight a lack of knowledge in users' feedback: the interest of users in the attributes of items is hardly captured.

Indeed, items contain many attributes, and moreover they may take several values (such as a comedy genre or a concrete actor). For simplicity, we call "features" all kind of information that may describe items (or users). We claim that the interest of users in items' features may enhance the accuracy of recommendations. We believe that recommendations taking into account the users' interest in the item's features are generally more accepted, as long as the user can easily recognize these liked features. For instance, users preferring a particular actor will better accept a movie recommendation where this actor is playing.

However, finding out the users' interest in the features of items is challenging. The large amount of features makes explicit feedback in features inappropriate. For instance, users hardly would rate every actor in a movie. In addition, this information is very large and heterogeneous, what may cause a complex integration into recommender systems as long as scalability issues and the increase of the domain dependency of the system.

Our main contribution is the definition and application of the implicit interest of users in items' attributes. This information is deduced by the number of times one has been in contact with an item's feature. Consciously or unconsciously. As a consequence, this number of occurrences is obtained by analyzing the existence of this feature among the whole set of past rated items. Thus, the higher number of occurrences the higher implicit interest of the user in the features.

Our goal is three fold: (1) to obtain the users' interest in the items' features implicitly, (2) to integrate this information into a recommendation process, and (3) to decrease the impact of the domain dependency in the recommendation process. For this purpose, we present a flexible and generic recommender system that relies on a collaborative filtering matrix factorization technique and implicit relations in data. We exploit the description of items to discover the implicit interest of the user in the items' features. The framework scores-up recommendations regarding not only the preferences of users in items, but also their implicit preference in the feature of the items. As a result, the most scored items

reflect the users' interest in items and in the features of the items. Indeed, we transform ratings into "semantic values", which better represent the general users' interests. The concept of semantic used here indicates the expansion in the meaning of ratings, rather than the presence of semantic technologies. That is, this semantic concept does not lead to inferences nor reasonings. A similar semantic concept was used in [Mobasher et al., 2004].

The experimentations use the well known Movielens and IMDb database, both publicly available. The results show the good performance of our approach compared to a standard matrix factorization approach. In fact, we particularly improve the precision and recall of this technique having little impact on the similarity among the items that are recommended.

This chapter is structured as follows: Section 4.2 presents the related work. Section 4.3 explains our approach. Section 4.4 gives the experimentations and results performed. Finally, we close this chapter by giving some discussions in Section 4.5.

4.2 Related Work

This section focuses on the state of the art of recommender systems dealing with explicit and implicit feedback, as well as the items' descriptions, in order to improve the recommendations. We are interested in which of these sources are exploited and how they affect the recommendation process. Particularly, we evaluate the related work in terms of scalability, heterogeneity and domain dependency of the systems.

Collaborative filtering techniques rely on the explicit or implicit users' interests in items. The most popular technique, Matrix Factorization (MF), has been very studied in the literature due to its accuracy and scalability. [Zhou et al., 2008] and [Schafer et al., 2007b] are matrix factorization techniques that only use users' ratings or users' implicit feedback (e.g. clicks). Recently, [Koren and Bell, 2011] have proposed a method to take into account both explicit and implicit users' interest in items under the same algorithm. Normally, matrix factorization techniques are scalable and some of them are widely implemented. However, despite being accurate, they do not simplify the incorporation of external heterogeneous data. Particularly, these techniques do not exploit the heterogeneous data which may enhance the recommendations (e.g. the features of users/items).

Pure content-based techniques make easier to deal with the items' features, although they are not as accurate as collaborative filtering and they have scalability issues. Thus, many authors focus on hybrid techniques. [Barjasteh et al., 2015] aim to insert to the matrix factorization technique the information given by the similarity between items' descriptions or users' descriptions, i.e. attributes. In fact, matrix factorization decomposes the rating matrix into two matrices: the users-features matrix, "P", and the items-features matrix, "Q". An item (or a user) is then defined by a vector of features, and hence it is possible to compute items' similarities. The authors propose to decouple this already built matrices ("P", "Q" or both) and make the features inside to evolve. These features change regarding other similarity matrices based on items' descriptions (or users' descriptions) in a phase called "transduction". As a result, this approach integrates external information into matrix factorization. In addition, this approach copes with cold-start: when users' ratings (or items' ratings) are not declared yet, the similarity between users' (or items) based on their description allows to correctly generate recommendations.

[Adomavicius and Kwon, 2007], [Li et al., 2008], [Lakiotaki et al., 2008], [Mikeli et al., 2013a], [Mikeli et al., 2013b] explain the concept of "multi-criteria" recommendations. This idea considers the explicit ratings of users as a solution for an equation where the variables are some items' attributes. These attributes are also linked to other explicit ratings, and thus it is possible to generate predictions in term of users' explicit interest in attributes. This helps to explain the overall rating value given by a user to an item. In addition, they change the concept of rating from an interest value representation to an ordinal representation. Thus, rating values of 5 and 4 stars are equally preferred to 2, etc. However, these approaches assume the existence of explicit ratings for the attributes of items. These ratings are hard to get in real-life since users are not willing to rate many items/attributes [Oard et al., 1998]. In addition, it is difficult to compute this approach in a distributed way.

[Liu et al., 2012] propose to detail users' interest in items as well. They suggest a three-layer representation, user-interest-item. For a user, an interest is a characteristic (a feature, such as an actor in a movie) that an item must have. For an item, an interest is one of its features. Then, they apply a Latent Dirichlet Allocation (LDA) algorithm based on "topic

models" from text domains (see [Blei et al., 2003]) in order to tackle the similar multiple "theme" problem. Hence, the authors interpret that the text documents are users, the words are items, and the topics are the (latent) interests. This approach extracts hidden users' interests by establishing a correlation matrix graph between items and interests. Despite its good performance, the complexity is not acceptable for large-scale applications.

[Ziegler et al., 2004] suggest a recommender system that exploits an item taxonomy to establish a common base in the items' descriptions. By using past users' ratings in items, this technique allows to compute users' similarities even when users do not share any common rated item. The main idea is to interconnect users through the common content of items in order to predict unknown users' ratings. This is an application of the "collaboration via content" paradigm in [Pazzani, 1999]. [Weng et al., 2008] is a very similar approach to [Ziegler et al., 2004] that use items' taxonomies (typically used for content classification) to find out the relations between users' preferences in items and the structure of the taxonomy as well. They verify that users who share similar item preferences may also share similar taxonomic preferences. This helps to cope with cold start and to understand the implicit preferences of users. This definition is similar to our implicit users' interest with a major difference: rather than interconnecting users ("users' collaboration") through items' description ("content") we interconnect items' descriptions ("items' collaboration") to find out the best items for users ("content").

The analysis of the users' interest in the items' attributes is a great challenge due to the lack of explicit users' feedback and the huge amount of attributes and the values these may take. This interest has not been successfully captured in the literature, although it certainly affects to the items that the user will choose in the future.

The main difference of our work to the current state of the art is the definition and use of the implicit interest of users in the items' attributes. In addition, we exploit the strengths of matrix factorization by adding an external and scalable layer, which allows to (1) distribute the analysis over the users' interests in the attributes of items, and (2) reduce the domain dependency of the system by separating the mentioned analysis from the collaborative filtering technique. We transform the input or the output of this technique in order to better adapt the recommendation for users. This reduces the complexity of the

system, integrates the new users' interests and improves the recommendations.

4.3 Architecture of our approach

The external heterogeneous information of users or items can enhance recommender systems [Peis et al., 2008]. The main issue is the impact of this data in the recommendation process. For instance, hybrid recommender systems based on collaborative filtering and content-based introduce more complexity, more domain dependency and decreases the scalability. We aim to achieve three goals to improve recommender systems: a capacity for incorporating heterogeneous information, a high level domain genericity and a scalable system. As a consequence, we aim to enhance the performance and accuracy of recommender systems.

In this work we take advantage of the domain independence and the scalability of matrix factorization. Our goal is to use it together with heterogeneous data, although reducing its impact in the recommendation process. As a consequence, we propose to separate the recommendation algorithm from the usage of the heterogeneous data. We dedicate an external layer, where all the users, items and items' features are analyzed together in order to find the implicit relations in data. The main task is to integrate this new data in such a way that standard matrix factorization can deal with. We introduce the concept of "semantic values" or "semantic ratings". Note that the term "semantic" indicates an expansion in the meaning of ratings. In fact, we transform ratings into "semantic values", and thus this new value represents the interests of the users in items and the attributes of the items.

We suggest a three-layer recommender architecture: a pre-analysis layer, a semantic layer and a recommender layer. Figure 4.1 shows our system's architecture. Since the number of attributes and the number of values for the attributes might be huge (e.g. all the actors in movies, or all movies' tags), the pre-analysis module implements a feature selection and a counting algorithm to quickly obtain the implicit interest of users in the selected features. The semantic module uses the information deduced in the previous layer in order to transform the ratings of users: we expand the meaning of ratings by adding the implicit relation in data. Finally, the recommender module contains the recommendation

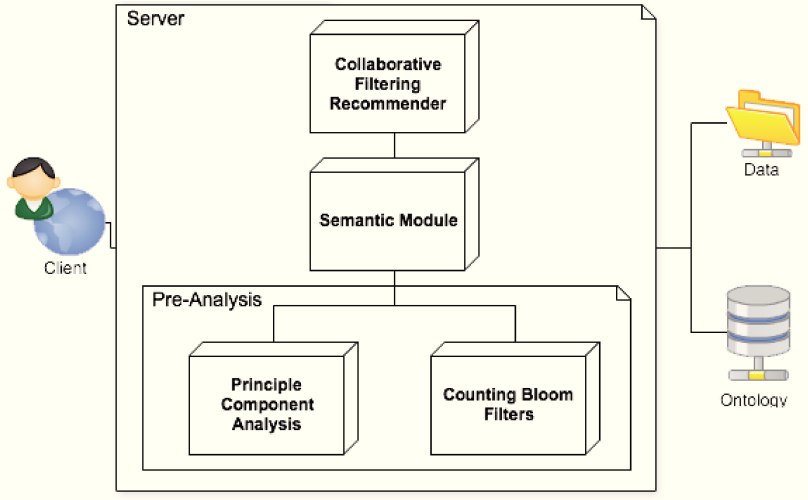


Figure 4.1: Global architecture of the recommender system.

algorithm.

4.3.1 Pre-analysis module

This layer deals with the information from the dataset and the domain description of items (e.g. a database or an ontology), making it abstract and quickly available for next modules. First, we study the relevancy of the attributes in the domain by using a feature selection technique based on Principle Component Analysis (PCA). Then, we analyze the interest of users in these selected attributes and store the deduced information in a fast and low space counting module called Counting Bloom Filter (CBF). This process is described below.

4.3.1.1 Feature selection

This module aims to obtain weights that measure the relevancy of attributes for users in the dataset. This task can be done by domain experts or exploratory analysis. As long as we aim to reduce the domain dependency of the system, we chose to perform an automated approach. In addition, the number of attributes can be large. As a consequence, feature selection based on reduction technique, such as Princinple Component Analysis (PCA), may accomplish both tasks.

We focus on how the variance of these attributes affect the rating of items. Thus, the variances are indicators (weights) of relevancy over the global dataset. We use them to balance the importance of attributes in the transformation of ratings into semantic values, and thus, in the recommendations.

4.3.1.2 Counting module

The typical ratings show the interest of users in items. It is important to understand their item rating-reasons in order to better serve the users. However, an item is composed of several attributes and getting feedback for all of them is complicated. Indeed, users are not willing to rate every single attribute of a movie. As a consequence, we suggest to implicitly gather this information using the past rated items. We count the number of occurrences of a given features among all the past rated items for a given user.

Databases or semantic technologies as ontologies describe the items' environment and they can easily return their unique properties. This fact gives free access to navigate through items' features, and thus we can find out the implicit interest of users easily. However, this task is slow for large number of users, items and features, and the number of requests to perform is large. As a consequence, this implicit interest should be computed off-line and stored in order to have it quickly available. In addition, this implicit information tend to be incremental. New rated movies carry out more implicit interest in features.

The counting module aims to find out the implicit interest of users in items and to store it in Counting Bloom Filters, which are bit-structures that represent "n"-elements of the same set "S" in a lower space of "m"-bits (see Annexe B.2 and B.3, or the paper [Broder and Mitzenmacher, 2004] for more details about bloom filters). This structure evolves from standard bloom filters by incorporating a separated bit-structure to count the number of repetitions of the same element "n".

The steps of this module are as follows:

1. For each user we create an empty counting bloom filter;

2. For each rated item by this user, we extract its features (i.e. all possible description of the item given by atributes values and other sources)

3. Finally we insert these features in the counting filter. Thus, the filter contains all the items' features which have had an interaction with the user. Highlight that each user has his own counting bloom filter, and these filters are used by the semantic module in order to improve recommendations.

4.3.2 Semantic module

This module aims to expand the meaning of ratings by incorporating the implicit interest of users in the attributes of items. As said above, an item is composed of several attributes and getting feedbacks for all of them is complicated. The counting bloom filter of a user contains the implicit interest of the user in the attributes of an item. We aim to exploit this information in order to add a new sense to the user's feedback.

The semantic module transforms the initial rating given by users into a new semantic rating value. Indeed, this new value takes into account not only the user preference in the item but also her/his preference in the attributes of the item. For instance, an item rated as 4 out of 5 may transform its rating value into 4.5. This fact reflects that this item has several attributes in common with the rest of items rated by the user. As a consequence, this boosts the recommendation of items which contain similar attributes to the ones the user liked in the past. Hence, recommended items are more suitable and acceptable by users because they may recognize relevant features for them.

The transformation of the ratings into "semantic values (sv)" follows the Equation 4.1. We call it "semantic equation".

$$sv_{u,i} = r_{u,i} + E[r_{u,*}] * \frac{\left|\sum_{j=1}^{F} C_j * W_j\right|}{N_u} \tag{4.1}$$

Here, $r_{u,i}$ is the real rating for item "i" given by user "u". N_u is the total number of items rated by user "u". $E[r_{u,*}]$ is the average of the ratings given by user "u". F is the number of selected attributes. W_j are the weights for these attributes computed by the feature selection module. C_j are the number of times that the value of an attribute has appeared for a user, easily retrieved by taking advantage of the computed counting module. Besides, since parameters are pre-calculated, the number of attributes does not

have a relevant impact on the execution time of the module. In addition, the process of this equation is easy to parallelize using a Hadoop/Map-Reduce paradigm [Dean and Ghemawat, 2008].

Moreover, we use this equation in two different levels of the recommendation. On the one hand, we apply it to all the ratings available in the original training dataset, which is the input approach. On the other hand, we apply the semantic equation to the output of the recommender system to modify the recommendations predicted by the standard collaborative filtering algorithm.

4.3.2.1 Semantic Dataset (input approach)

This approach implements the semantic module at the input of the recommender module. Briefly, it transforms the feedback (i.e. ratings) in the training dataset into a semantic feedback, according to Equation 4.1.

For each rating a new "semantic rating" is computed. Hence, a "semantic dataset" is built from the original one. Figure 4.2 shows this approach. The semantic module takes a training dataset, which contains the "original dataset", and generates a new "semantic dataset", which contains the new "semantic ratings". This latter is used to train the recommender module and create a prediction model to exploit. As the incoming dataset has changed, the recommendation module returns different items.

Collaborative filtering finds out patterns in the users' ratings in items to group users with similar preferences. Our approach scores up items depending on their features and the interest of users in these features. This has two main consequences over the predictions given by the system. On the one hand, items are still chosen by seeking patterns in the users' feedback. On the other hand, predicted items suit the users regarding not only the community of users with similar preferences but also the features that these recommended items contain. We involve not only items but also attributes and features. In fact, by increasing the ratings of items in which users are interested (according to their interest in the attributes of items), one helps the recommendation technique to focus on such accuracy and predictions.

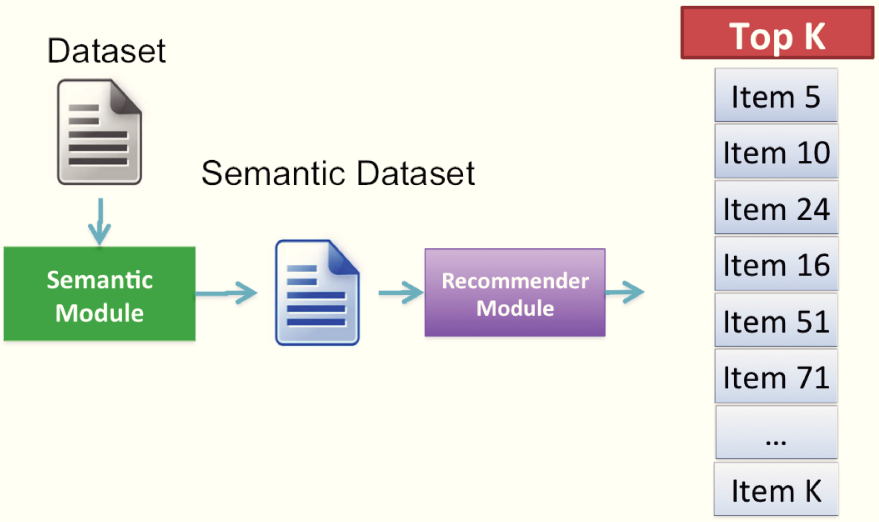


Figure 4.2: Semantic Dataset: input approach

Table 4.1: Dataset example.

Table 4.2: Users and items' ratings.

User	Movie	Rating
1	1	4.0
1	2	3.0
1	3	1.0
1	4	2.0
2	9	4.0

Table 4.3: Attribute actor. Items' actors.

Movie	Actor
1	Actor 1
1	Actor 3
2	Actor 1
3	Actor 2
4	Actor 1
10	Actor 3

Table 4.4: Attribute genre. Items' genre.

Movie	Genre
1	Comedy
1	Fantasy
2	Comedy
2	Drama
3	Thriller
3	Drama
4	Comedy
4	Fantasy
10	Comedy

4.3.2.2 Example

Tables 4.2, 4.3 and 4.4 show an example of a movie dataset which contains information about users' ratings in items and the attributes of these items (actors and genres). This approach takes and modifies every rating in the dataset according to the implicit interest of users in the attributes of items.

Let's focus on the rating of the item 1 given by the user 1 ($r_{u,i} = r_{1,1} = 4$). Our goal is to obtain a new "semantic rating". We first calculate the average of ratings for this user,

who has rated $N_u = 4$ movies:

$$E[r_{1,*}] = \frac{4.0 + 3.0 + 1.0 + 2.0}{4} = 2.50 \tag{4.2}$$

Secondly, we get the weight for attributes computed by the feature principle component analysis (e.g. $W_1 = 0.4$ and $W_2 = 0.6$ for genres and actors respectively). The third step is to get the implicit occurrences stored in the counting bloom filter:

- The user 1 has rated the items 1, 2, 3 and 4, and these items have actors and genres.
- Focus on the item 1 and its genres: comedy and fantasy. The movies 2 and 4 already rated by user 1 are comedies. Besides, the movie 4 is also a fantasy movie. Hence, the occurrences count is $C_1 = 3$.
- Focus on the item 1 and its actors: actor 1 and actor 3. The actor 1 also appears on movies 2 and 4. Thus, the occurrences count in this attribute is $C_2 = 2$, because the actor 3 does not appear on any other movie.

Putting everything into Equation 4.1, we obtain the new "semantic rating":

$$sv_{1,1} = 4.0 + 2.50 * \frac{|3 * 0.4 + 2 * 0.6|}{4} = 5.5 \tag{4.3}$$

4.3.2.3 Semantic Top-K (output approach)

Collaborative filtering are good techniques to guess the interest of users in items. However, recommender systems usually present a top-K items to users. This top-K is typically ordered by the predicted interest value for users, hence the first item is likely preferred to the second one, etc. The user is free to pick up any of these K items. Our approach in this context aims to re-order this top-K (and eventually make new items appear) to present first the items containing the most relevant features for the user. As a result, by using the semantic module at the output (i.e. predictions) of the recommender system we adapt the recommendations to users based on his implicit feedback in the features of items.

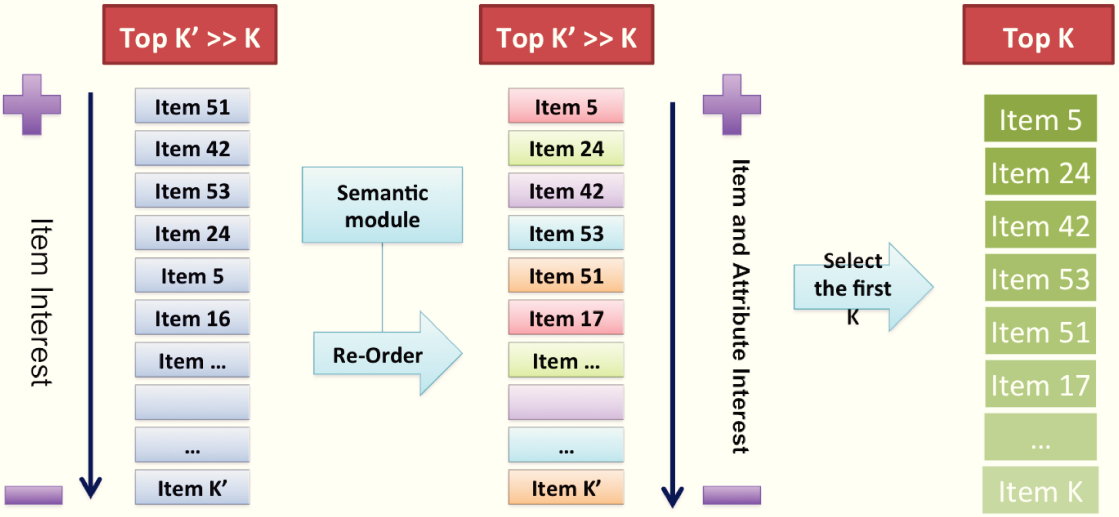


Figure 4.3: Semantic top-K: output approach

Figure 4.3 represents this approach. The recommender system gets a top-K' ($K' >> K$) best predictions to present to a user u. The semantic module captures this top-K' and transforms every rating prediction of these recommendations into "semantic predictions". For this purpose, two sets are required: (1) the recommended items in the top-K', and (2) users' ratings, i.e. the set of rated items by the user. The former is the set of items to transform. The latter implicitly contains the attributes in which the user is more interested.

By computing the semantic prediction for every item in the top-K' we obtain a new resorted top-K', which contains the same items in different positions. Thus, we have scored up items with similar attributes to the ones the user is interested in. Finally, we return and present to the user only the top-K items of this new resorted set. The fact of taking $K' >> K$ helps the system to put in the top-K new relevant items which initially were out of it.

This approach is much faster than the semantic dataset because it requires to transform many less ratings. In addition, since collaborative filtering uses to return a certain grade of diversity in their predictions, we adjust the top-K items according to the interest in items and attributes.

Table 4.5: Example. Top-3 recommendations for the user "1"

Top-3	Movie 21	Movie 10	Movie 64
Predicted Rating	5	4.5	4

4.3.2.4 Example

Again, we use Tables 4.2, 4.3 and 4.4 to represent our dataset example. In addition, we use Table 4.5 to represent the top-K recommended items for a user (user "1"). The goal is to modify the predicted ratings in the top-K by using the Equation 4.1. We already know that $E[r_{1,*}] = 2.50$, $W_1 = 0.4$ and $W_2 = 0.6$. The current value to modify is one of the predicted recommendations, for instance the prediction of movie 10 ($r_{u,i} = r_{1,10} = 4.5$). Now we get the implicit occurrences stored in counting bloom filter:

- The user 1 has rated the items 1, 2, 3 and 4, and these items contain actors and genres.
- Focus on the item 10 and its genres: comedy. The movies 1, 2 and 4 already rated by user 1 are comedies. . Hence, the occurrences count is $C_1 = 3$.
- Focus on the item 1 and its actors: actor 3. The actor 3 appears on movie 3. Thus, the occurrences count in this attribute is $C_2 = 1$.

Putting everything into Equation 4.1, we obtain the new "semantic rating":

$$sv_{1,10} = 4.5 + 2.50 * \frac{|3 * 0.4 + 1 * 0.6|}{4} = 5.625 \tag{4.4}$$

Applied to the whole top-K', this process creates a new order in the top-K. These new recommendations are more personalized to the user according to the interest in the attributes of items.

4.3.3 Recommender module

The recommender module contains the recommendation algorithm, which analyzes the ratings or "semantic values" in order to generate recommendations. Our architecture uses a standard matrix factorization approach.

4.4 Experimentations

We suggest to use the GroupLens dataset proposed by [Cantador et al., 2011], which merges ratings in MovieLens dataset[1] and movies' attributes in IMDb[2] database. The dataset is composed of 2113 users, 10197 items and 855598 ratings within a scale of 1-5. It offers six attributes: genre, directors, actors, countries, locations and tags. It has 112881 different attributes' values (i.e. features): 20 movie genres, 95321 actors, 72 countries, 4266 locations and 13222 tags.

We first explain the module settings (feature selection, counting and recommender modules) and second we show the context and evaluations of our experimentation.

4.4.1 Module settings

4.4.1.1 Principle Component Analysis

This step aims to (1) reduce the high number of attributes (if required), and (2) compute a weight to represent the relevancy of attributes in the semantic equation.

We use a subset of ratings. We extract the 100 users who have rated the highest number of movies and we obtain 169155 ratings, which represent almost the 19.77% of the total ratings in MovieLens dataset. This module analyzes the relevancy of items' attributes over this data and returns the most representative features. In addition, it returns ceiled-weights for these attributes based on their contribution to the feature selection model. Table 4.6 shows the results of the feature selection analysis, as long as the attributes' variance contributions to this analysis. The two main dimensions in which the dataset can be plot are composed mainly of actor, country, director, genre and location, and thus the tags attribute contribute very less. We decided to avoid using this attribute "tags" in our experimentation.

[1] http://grouplens.org/datasets/movielens
[2] http://www.imdb.com/

Table 4.6: Experimentation: Weights % for variables in dimensions. Approximate values.

Variables	actor	country	director	genre	location	total
Dimension D_1	19.537	12.719	19.896	0.000	5.064	57,25
Dimension D_2	4.785	8.3732	5.303	6.459	17.823	42.75
Total (%)	24	21	25	6	23	100

4.4.1.2 Counting Bloom Filter

The counting bloom filters are built in off-line in order to speed up the semantic equation. The dataset contains 2113 users and 112881 different values for the attributes. Regarding the parameters of the bloom filter we consider to accept a very low false-positive ratio of 0.01%. In addition, we set the counter parameters to 6 bits, and thus, the maximum number of appearances of a feature for one user is set to 64 times. As a result, the size of one filter corresponding to one single user is around 1.3 Mb. Hence, for the 2113 users the total size of all filters is around 2.7 Gb. This space consumption allows to increase the speed of the semantic module.

4.4.1.3 Recommendation technique

We use a matrix factorization algorithm in Apache Mahout[3] which uses a gradient descent optimization technique to solve the matrix factorization paradigm and to build the recommender core. This algorithm will iterate a maximum of 30 times to find out the best 30 latent space dimensions that better explain the ratings. However, the semantic module uses this recommender as a black box, either modifying the input or the output of the recommendation algorithm.

4.4.2 Evaluation

Our baseline is a pure collaborative filtering technique based on matrix factorization. It is compared to our approaches, which use the same technique as the core of recommendations to predict ratings. Our system analyzes and obtains the users' implicit interest in the items' attributes in a layer separated from the collaborative filtering technique. This

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

reduces the domain dependency of the system and enhance collaborative filtering recommendations.

We do not implement any content-based technique to deal with the implicit interest of users in the attributes of items. Indeed, our approaches analyze and obtain this information separately in order to enhance collaborative filtering recommendations. This reduces the domain dependency of the system. Due to this assumption, we do not consider our approach a hybrid method: the core of recommendations remains a pure collaborative filtering technique.

Our system relies on the analysis of the training data. The more users' ratings (higher training datasets), the better one can find the implicit interest of users. Therefore, for the evaluation of the systems we use the full MovieLens dataset containing 855598 ratings over 10197 movies. In order to represent different training sets (i.e. sparsity levels), we randomly split the dataset into 90%, 80%, 70%, 60% and 50% training sets. The remaining percentage in each level is the test set[4]. As a consequence, we can train systems and compare the predictions in the model with the real-observed values in the test set.

We evaluate our approach under four contexts. First we aim to show an outline where recommendations given by our approach are pertinent and likely to be accepted by users. We develop an example using a user in the dataset. Second, we aim to measure the performance of recommendations using three different evaluations: a rating prediction accuracy, a ranking accuracy and a top-K intra-similarity evaluation. Finally, the charts show the results of three approaches: (1) the standard matrix factorization which is denoted as "mf", (2) the implementation of the semantic model at the input, denoted as "semantic dataset", where matrix factorization ingests semantic ratings, and (3) the implementation of the semantic module at the output of the matrix factorization predictions, denoted as "semantic top-K", where the semantic module transforms predicted ratings.

[4]Denote that, since the convergence of the collaborative filtering has been already proved and the semantic approaches do not modify this convergence capability, we do not need a cross-validation set.

4.4.2.1 Illustrative example

This section aims to give a deployed example of what the recommender systems return. It visually compares top-K returned items from the different approaches. The interest of this outline is to compare the items that different recommenders may show to the same user.

The example focuses on the user 6757 due to his large number of ratings. This user has evaluated 119 items with the maximum rating score (ratings equal to 5). We extract 60 out of this 119 from the training set. Our goal is that the recommender system predicts some of these taken movies.

We train the three systems under this context. The matrix factorization creates a model using this training set, the semantic dataset first applies the semantic equation to the training set and then creates a model, and the semantic top-K approach modifies the recommendations returned by the standard matrix factorization. Finally, we ask the systems to retrieve a top-60 items for user 6757. It is expected that in this top appears some of the 60 before extracted items. Table 4.7 shows a shorter top-10 items (over these top-60 movies) and Figure 4.4 shows the movies behind IDs, which visually compares the top-10 movies. The standard matrix factorization returns 2 items (858 and 912) which belong to the extracted items. However, the semantic approaches improve this fact: the semantic dataset returns 4 items (858, 912, 1213 and 1221) and the semantic top-K returns 3 items (858, 912 and 1221). In addition, we notice the appearance of different items in the semantic approaches (such as item 3462). Especially, we highlight a new order in items of the semantic top-K (items 912 or 2624). In fact, we have scored up items which contain interesting attributes for the user, and thus, less interesting items regarding attributes get down in the list. These results show our assumptions: by adding the implicit interest of users in items, recommendations are more suitable and acceptable to users, i.e. more items out of the extracted high scored items set are predicted.

Table 4.7: Experimentation: Recommended Top-10 movies for user 6757. Items ID and predicted values.

MF		Sem. Dataset		Sem.top-K	
ID	Value	ID	Value	ID	Value
6669	4.34	6669	4.21	6669	4.40
26350	4.20	858	4.14	912	4.31
858	4.189	912	4.13	858	4.239
912	4.186	26350	4.09	8492	4.237
8492	4.16	7749	4.08	26350	4.23
7762	4.128	1221	4.07	3462	4.226
3077	4.1241	3462	4.05	2624	4.224
7749	4.1240	7762	4.03	4806	4.219
4806	4.12	1213	4.027	1221	4.218
2624	4.11	8492	4.026	7256	4.210

Figure 4.4: Experimentation: Recommended Top-10 movies for user 6757. Visual comparison.

4.4.2.2 Root Mean Square Error (RMSE)

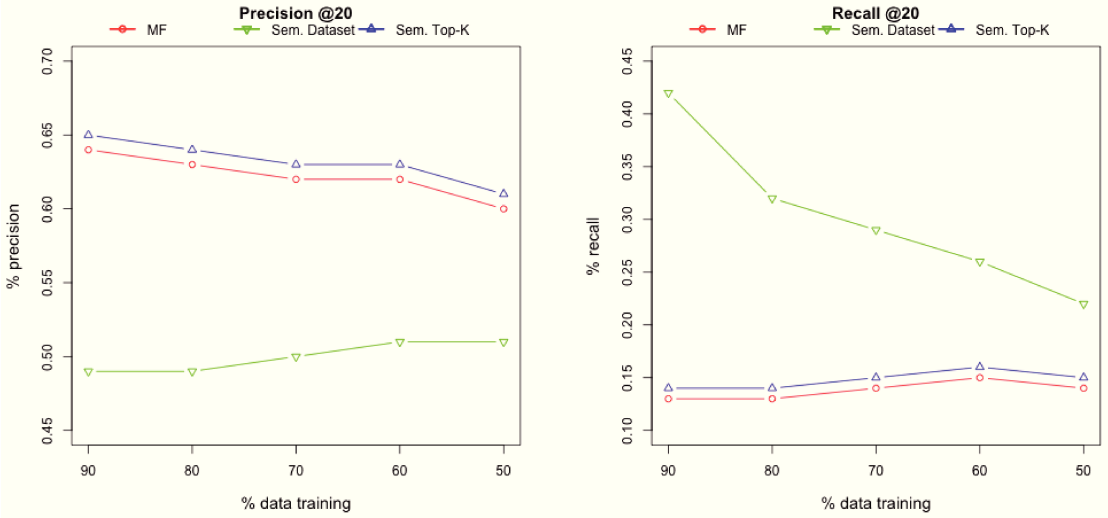
The RMSE measure evaluates the system in terms of accuracy of the ratings prediction. It represents the standard deviation in the error of the prediction. This error is the difference between predicted values and real-observed values in the test set. Thus, the lower is this error, the better is this metric.

The semantic module scores up items due to the presence of attributes, yet it does not penalize the absence of them or it does not penalize the presence of non desirable attributes. Thus, the semantic rating is always higher than the explicit ratings. This yield indeed to a change of rating's scale. As a result, our framework does not overcome this measure. In fact, our techniques are not adapted to outperform individual rating predictions due to the rating's transformation. Future work consists on improving this metric as well. In fact, the semantic module only scores up items regarding the positive interest of users in the attributes of items, although it does not score down items regarding a negative interest. This fact, together with a normalization within a ratings scale may improve this measure. This perspective is explained in Chapter 7.

4.4.2.3 Precision, Recall and F-Measure

Precision and Recall techniques measure the relevancy of items in a previously selected top-K. This relevancy is a binary value associated to the item: an item is relevant if the rating/prediction overpass a certain predefined threshold, i.e. to be greater or equal to 4-5. Precision represents the percentage of relevant items (items that should be recommended first) over the recommended top-K items. Recall represents the percentage of relevant items over the whole set of items. Figures 4.5 show the results in precision and recall measures of a top-20 recommended items. Note that results are the average for all users top-20 precision and recall.

On the one hand, the precision is high in matrix factorization due to the good RMSE. It easily puts relevant items in the top-K. However, the semantic top-K approach slightly overcomes this precision, since it scores-up items and thus other relevant items are likely to appear (making less relevant items disappear from the top). On the other hand, our



(a) Precision.

(b) Recall.

Figure 4.5: Precision and Recall metrics comparisons regarding a top-20 items.

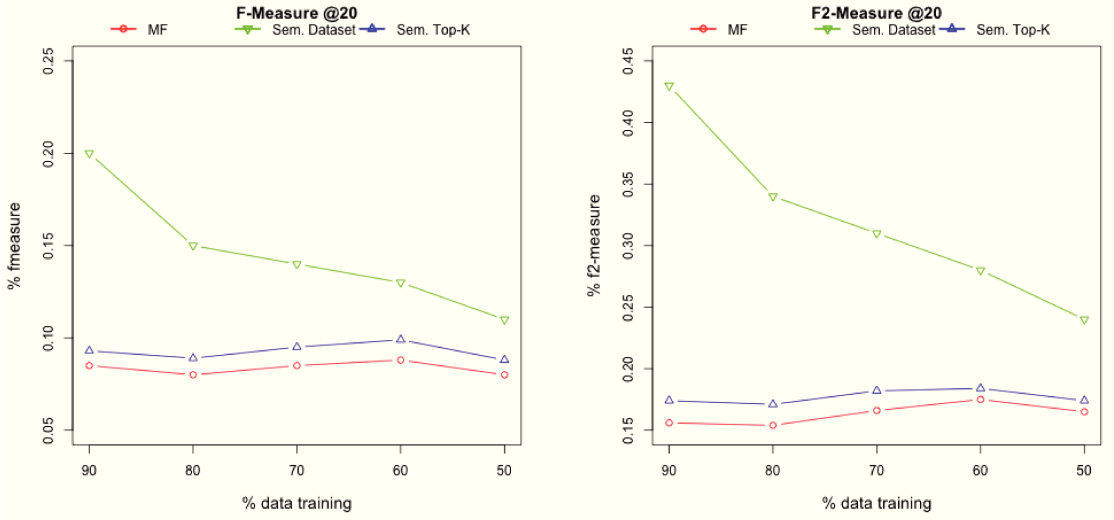
semantic approaches identify more relevant items among the whole dataset, and hence recall metric are higher, especially in the case of "semantic dataset".

F-Measure and F2-Measure are figure of merits for Precision and Recall. The former equally balance the importance of precision and recall. The latter gives the double importance to precision than to recall. Figures 4.6 show these metrics. One may observe that adding a semantic layer improves top-K recommendations, enhancing the overall performance of the system as well. Summing up, scoring up items with common attributes indeed increases the probability of taking relevant items.

4.4.2.4 Intra-List-Similarity (ILS) and Intra-List-Diversity (ILD)

Content-based techniques tend to recommend very similar items to users, i.e. over-specialisation. On the contrary, collaborative filtering methods introduce some diversity. In fact, recommending always too similar items may bore users, and too different items might generate confusion.

The ILS (Intra-List-Similarity) metric, also called ILD (Intra-List-Diversity), balances items similarity/diversity among a recommended top-K. In a scale 0-1, the closer is the



(a) F-Measure. Equally balanced F-Measure.

(b) F2-Measure. Heavy precision F-Measure.

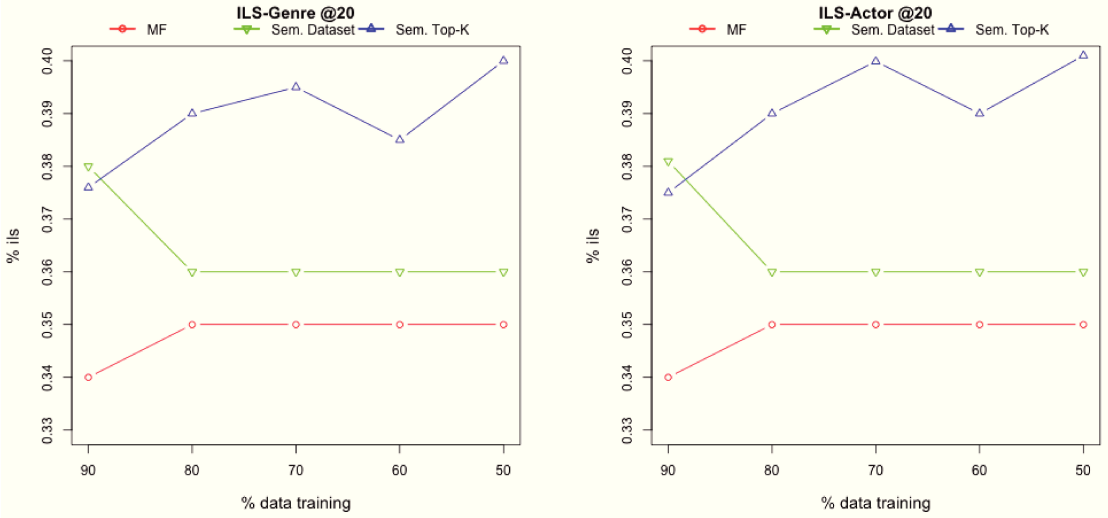
Figure 4.6: F-Measures for top 20 items.

value to "1", the more similar items in the top-K are between them. On the contrary, the closer to "0", the more diversity exists among recommendations.

Our framework exploits the description of items and thus it is expected that it increases the similarity among recommendations. Figures 4.7 represent the similarity/diversity measure regarding the genre attribute and the actor attribute of items. One may observe that semantic approaches yield to more similar top-K recommendations. However, this similarity is not only item oriented, that is, similarity is not only among items. In fact, there is a similarity between the items' features and the features that the user is interested in. As a consequence, the top-K recommendations are specially adapted for users: they may find diverse items, but items they can recognize and like due to the item's attributes.

4.5 Discussion

Recommender systems aim to personalize the web content for users. They analyze users' feedback in order to predict future users' interests. However, the interest in the attributes and features of these items is not captured. We believe that this interest may be useful to enhance recommender systems since it allows to adapt recommendation in



(a) Attribute "genre".

(b) Attribute "actor".

Figure 4.7: ILS metric comparisons.

details. On the contrary, this information is hard to retrieve: users are not willing to rate all aspects of items (e.g. all the actors in a movie).

We proposed an approach which relies on collaborative filtering techniques and implicit relations in data. Our goal is to enhance the accuracy in recommendations in order to render recommended items more acceptable by users and to reduce the impact of domain dependency in the collaborative filtering recommendation process.

On the one hand, the description of items allows making more relations among data. Thus, one can easily extract the implicit interest of users in the attributes of items. Using this information, we suggest to score up ratings given by users in order to represent also the implicit interest in the attribute of items. We called this new interest representation "semantic value", because it expands the meaning of ratings.

The presented architecture is divided in independent layers and allows a flexible usage. Two approaches are presented regarding this architecture: Semantic Dataset and Semantic Top-K. Both approach rely on a collaborative filtering algorithm based on matrix factorization. The former acts in the input of the recommender system by analyzing the whole train dataset and transforms the input ratings. By enhancing and scoring up items for which

the users have an implicit interest in their features, we help the collaborative filtering to focus on such kind of items and interests.

The latter aims to apply the semantic layer in the output of the system. Typically, recommender systems provide top-K items ordered by predicted user's preference. This approach better adapts top-K to users' especial interests in items' attributes.

The experimentation uses MovieLens dataset and IMDb database. The results show the performance of the approach over different measures. Specially, our approaches enhance the fact of taking relevant items for users. Thus, users might be more likely to click on recommendations because they may contain features they know and they are interested in.

Finally, in this chapter we have proposed to use the implicit interest of users in the items' attributes to enhance the accuracy of rank-based recommendations. The scalability issue has been addressed as well, the presented approach can be parallelized among multiple machines and the distribution of the recommender core has been explained in Chapter 3. In addition, this chapter has allowed to define users through a large quantity of interests and represent it into a bloom filter. The next chapter (Chapter 5) focuses and exploits the bloom filter representation of items and/or users to reduce memory consumption in large items'/users' descriptions.

Chapter 5

Coping with large vector representations of users and items in very large datasets.

"If you don't understand the details of your business you are going to fail."

- Jeff Bezos (Founder and CEO of Amazon.com)

This chapter is mainly extracted from the conference paper: *"An Item/User Representation for Recommender Systems based on Bloom Filters"*; Manuel Pozo, Raja Chiky, Farid Meziane and Elisabeth Métais; RCIS 2016.

5.1 Motivation

The recommender systems exploit the users' feedback in items to predict future users' interests. In addition, content-based or hybrid methods, use the items' and/or users' attributes, a.k.a. features, to enhance the correlation among items/users. These features can be very heterogeneous. For instance, in the domain of movies one film (considered as an item) can be described by its genre (e.g. romantic drama comedy), by the actors who play in (main actors, secondary actors), and by the directors and staff who participated in the project. Furthermore new tendencies add data from contextual and external sources, such as locations, open data, Wikipedia, Twitter and Facebook, in order to better describe items/users [Peis et al., 2008, Kantor et al., 2011, Dahimene et al., 2014]. However, this could create very large heterogeneous descriptions and render computations more difficult to handle.

The most known recommendation techniques are collaborative filtering, content-based and hybrid methods [Kantor et al., 2011]. Especially content-based and hybrid recommendation techniques deal with the above mentioned challenge. They typically represent items/users as a vector of features or keywords, and thus, vector similarity methods can be computed [Kantor et al., 2011]. The accuracy of the similarity between items/users will depend on the selected features to compare and will affect the recommendation. For example, comparing two movies only by their genre, such as comedy and drama, is less accurate than involving also their actors in the similarity process.

Indeed, the number of features to describe items/users affects the correlation of data. Generally the more attributes are used the better the accuracy of the similarity is. For example, two movies may share staff, genre and main actors, although they do not have common secondary actors. In a recommender context it may happen that one user is especially sensible to a particular actor, what has an effect (positive or negative) in the user's interest. This sort of particularities are skipped in recommender systems due to the short items/users descriptions.

On the contrary, one may notice that large resources contribute to large item/user descriptions, thus making the representation difficult to deal with in terms of sparsity and space-consumption. In fact, new features may not appear in other items, for instance, a new actor that has played only in one movie, and increasing the number of features increases the size of vectors that will eventually become larger and more sparse. Furthermore, increasing vector sizes makes similarity computations slower.

Current data analysis systems (e.g. expert systems and feature selection models) reduce data representation into a lower space in order to reduce both the sparsity and space complexity. However, reducing features may yield to a loss of quality in the item representation, and consequently, in items similarity. Besides, the enormous size of the datasets involves big memory resources and a very large time analysis while building these models.

In this chapter we deal with the representation of items/users. Our goal is to represent items/users by a high quantity of features in order to have a great detailed description. This makes the similarity process in recommender systems more accurate. In order to cope with this big representation, sparsity and space-consumption issues, we propose to

use a compressed yet high quality data representation for items/users based on Bloom Filters [Broder and Mitzenmacher, 2004]. A Bloom Filter (BF) is a bit structure that allows to represent a set of elements in very low space. To the best of our knowledge, these filters have not been used for items/users descriptions in the field of recommender systems. The motivation and the three following contributions are applied to a recommendation process:

1. Bloom filters highly reduce the size of item representations while having a great number of features. This fact involves a detailed items/users descriptions in a lower space complexity;

2. Bloom filters allow a fast bitwise AND operation to compare common features of items that we use to compute an accurate similarity measure;

3. Items' and users' similarities are not only in common features, but also in common missing features. To address this issue, a bitwise XNOR similarity operation is proposed and takes into account both common features, and common missing features.

In our experimentations we use the MovieLens dataset and IMDb database, which are publicly available [Cantador et al., 2011]. The tests have been performed for: (1) analysing the pertinence of bloom filters in recommender systems, (2) enhancing the usage of a bitwise AND similarity instead of the common vector cosine/jaccard similarities and (3) using a bitwise XNOR similarity. We compare our bloom filter item representation against two models: a vector similarity model that uses all available features and a dimension reduction technique based on Singular Value Decomposition (SVD). The results show that our approach highly reduces the size of vector representations (97% per vector) while keeping a high fidelity in item similarity (accuracy of 98%). In addition, it outperforms the results of SVD.

The structure of this chapter is as follows: Section 5.2 gives the related work in similarity processes and the representation of items/users applied to recommender systems. In Section 5.3, the proposed similarity model based on bloom filter representations is explained in details. In Section 5.4, the experimentation phase and the achieved results are

presented. Finally, we discuss the researches in this chapter in Section 5.5.

5.2 Related Work

This related work focuses on researches especially using a similarity computation process in recommender systems. Particularly, we focus on the representation of items/users in these systems, the resources which they aim to exploit and how they deal with description complexity.

On the one hand, similarity-based collaborative filtering methods represent items/users as a vector of feedbacks, as stated in [Breese et al., 1998, Sarwar et al., 2001, Lemire and Maclachlan, 2005, Su and Khoshgoftaar, 2009]. Then in order to find out similar items/users, correlations and similarities among these vectors are performed. The model-based collaborative filtering techniques make this vector representation obsolete [Hofmann, 2003, Su and Khoshgoftaar, 2009, Koren, 2009, Koren et al., 2009, Koren and Bell, 2011]. The already explained matrix factorization family belongs to this group. These methods build a model which represent the items and users feedback in a lower space rank. However, typical model based techniques solely rely on the interest of users in items. Particularly, they do not exploit other resources such as the domain of recommendation (e.g. movie's domain) or the general context of the recommendation (e.g. localization, users' affordability).

In general, collaborative filtering has already demonstrated simplicity and better accuracy than content-based, although hybrid methods may enhance these techniques by incorporating heterogeneous data. In content-based techniques, items are represented by a vector of keywords/features [Tiroshi et al., 2012]. Hybrid methods may combine both techniques to get stronger performance. Consequently, they also exploit the features of items/users and may use a vector to represent them.

The similarity process in recommender system helps to group and classify items and users. [Boim et al., 2011a, Boim et al., 2011b] propose a framework to control the similarity (versus diversity) factor in a top-K recommended items. They create "trees of interests" that allow creating a "zoom-in" technique to see more items of the same tree, which tend

to be similar. Thus, the better is this similarity accuracy the better is possible to group similar items. [Peis et al., 2008] present other tendencies in recommendations that use semantic technologies, like ontologies, to better describe users and items [Peis et al., 2008]. For instance, [Fernández et al., 2006] and [Pan et al., 2010] propose a hybrid method to describe users and items through ontologies structures in order to compute an inference similarity. Again, the better is the ontology description the better is the inference extracted.

Other authors propose to incorporate more heterogeneous data from external resources. [Katz et al., 2011] use item-item similarity based on the context in Wikipedia pages to compute "artificial ratings" for an item. [Werner et al., 2013] characterize economic articles using attributes as keywords. Then, they compute articles cosine vector similarity regarding their keywords. Finally, articles that match better with a user profile are recommended.

The description of items can be very large and sparse due to the numerous and diverse resources, e.g. movies genre, actors, directors, writer, locations and social network tags. To deal with such quantity of features, some authors propose to perform a features selection which can be supported by domain experts or by using explanatory analysis, such as topic modelling, Singular Value Decomposition (SVD) or Principle Component Analysis (PCA).

For instance, [Vozalis and Margaritis, 2007] propose to add demographical and context information to the recommendation. They create users and/or items demographic vectors in order to represent information. If the dimension of these vectors is too high, they reduce it by applying the SVD technique. A cosine similarity method is later used to compute user and item similarities. [Mobasher et al., 2004] suggest a hybrid recommendation technique based on content-based item-similarities and an item-based collaborative filtering technique. This similarity uses semantic sources to exploit the attributes of items. They create an item-attribute matrix, but it may be too noisy and attributes may be correlated. Thus, the authors reduce it by using SVD and compute items similarities.

One may observe that most similarity processes do not take all possible or available attributes information into account. The item similarity measure might suffer from a loss of accuracy since less features have been taken into consideration for this comparison. Although larger number of features may improve similarity, the computation might become more expensive. In addition, approaches dealing with space reduction or latent space

models may lose information.

We propose to use a representation of items/users based on bloom filters. This reduces the size of vector representations and compute accurate and faster similarities. [Bloom, 1970] introduced the concept of bloom filter and many research fields and applications have taken benefit of this structure due to its memory-efficiency and fast-capabilities [Broder and Mitzenmacher, 2004], for instance databases queries or computer networks. Recently, [Geravand and Ahmadi, 2014] applied these filters to create a fast, accurate and private plagiarism system that compares the similarity among documents of different databases. [Jain et al., 2005] used them for web search field. Typically web browsers return to users' request top websites which tend to be very similar. The authors propose to group similar websites results in order to allow the return of other diverse sites in the top results.

To the best of our knowledge, the only reference of bloom filters in recommender systems is in [Pagare and Shinde, 2013], where the authors apply this structure to improve a parallelization MapReduce paradigm rather than to enhance any part in a recommendation process. Hence, this structure has not been used yet in the core field of recommender systems.

5.3 Bloom Filter Similarity Model

The Bloom Filter Similarity Model, a.k.a. Bloom Model, aims to alleviate the aforementioned issues. This approach relies on bloom filter representations of items/users. By exploiting bloom filter's properties, it is possible to compute similarity measures to know the similarity among two filters, and thus, among the items/users theirselves.

In this chapter, we consider that the reader is already familiar with bloom filters, and particularly, with the mathematical and probabilistic properties behind (i.e. near optimal false positive, intersection and union). However, Annexe B explains the concept of bloom filters. Table 5.1 resumes the notation we use in this chapter. In addition, comparisons and references to the simple vector similarity model are also given.

In order to define the context our bloom filter similarity model, let S_N be the set of all possible items features in a database used in a similarity comparison, so that $|S_N| = N$ is

Table 5.1: Notation used for bloom filters

Notation	Description
BF	Bloom Filter
n	Number of expected insertions
m	Size in bits
k	Number of hash functions
fp	false positive

the number of features in this set. Moreover, let S_i be the set of active features of an item "i", and $|S_i| = n$ the number of features in this set. As a consequence, $S_i \subset S_N$. In addition, within this context we assume that $n \ll N$.

For instance, "N" can be the total number of actors, directors, editors and tags in a movie database. However, a single movie only has "n" of these features, a.k.a. active features. Normally increasing the features in the database has more impact over "N" than over "n". Indeed, incorporating external information increases the value of N, but only few items will increase their active entries "n" significantly. Thus, vector representations are larger and sparser.

Under this context and definitions, the state of the art proposes (1) to represent items/users as vector of attributes/keywords/values, and (2) to reduce the size of vectors by using some feature selection criterion. However, this may lead into large vectors or a loss of description accuracy. We suggest the bloom filters to represent items/users as a set of features. Indeed, these structures are low-size hashed binary vectors representing a huge set of elements. Therefore, we represent *one item by one bloom filter* which contain all the "n" active features for this item. Thus, the number of expected insertions is equal to the number of active features for one item only, and items can be compared by using their associated filters to compute items' similarities. Figure 5.1 represents the difference between the vector similarity model and the bloom model.

Our goal is to analyse the impact of different bloom filter settings in an item similarity process. We first detail the necessary conditions for creating a bloom model. Second, two similarity measures are presented: AND bitwise and XNOR bitwise operations. We conclude this section by presenting the consideration of the similarity measures in rec-

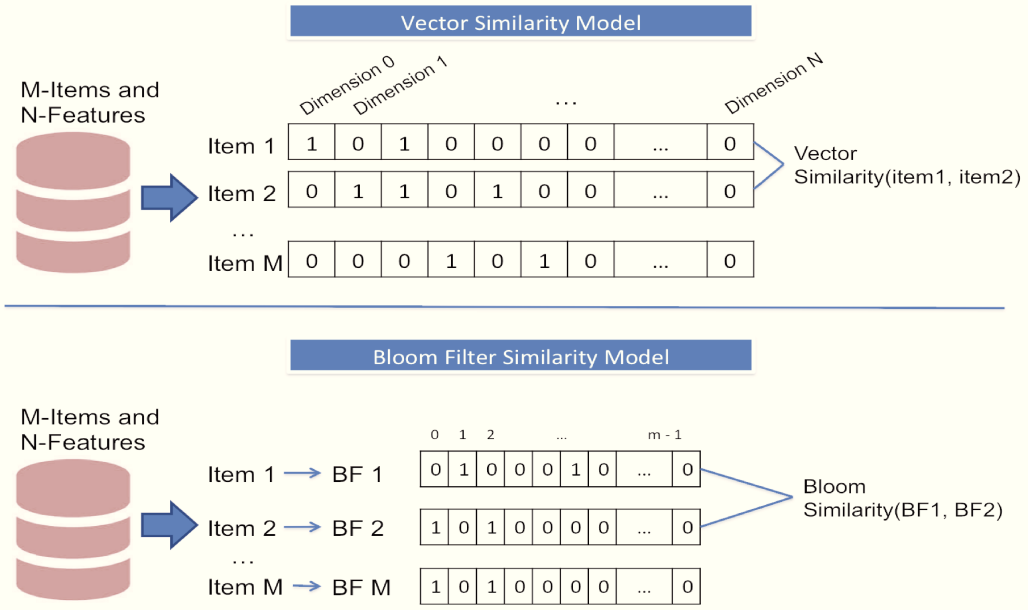


Figure 5.1: Vector Similarity Model versus Bloom Filter Similarity Model. One item is represented by one vector/filter. Items' similarities are computed by using these vectors/filters.

ommender systems, the existing trade-off in our bloom model and the short-comings and solutions of this approach.

5.3.1 The Bloom Model Condition

Bloom filters have four parameters: the false positive ratio fp, the number of hash functions k, the size of the bit-structure m and the number of expected insertions n. The only condition to perform operations over two bloom filters is to have the same size m and hash functions k. However, we consider a new condition to take into account to build the bloom similarity model.

On the one hand, items and features are known a priori in a recommender application (the dynamic and incremental situations are later introduced in this section). In this case, as one item is represented by one filter, the number of expected insertions (i.e. active features) for this item is also known and represented as n. On the other hand, to perform bloom filter comparisons the filters have to share the same hash functions k and the same size m. Thus, given n and fixed m and k: (1) the false positive fp increases in each

insertion of an element, and (2) the maximum false positive fp_{max} is given by the maximum insertions n.

However, the items do not have the same expected number of active features. As long as m and k have to be equal for every filter, the predefined number of insertions should be fixed as well to achieve the maximum desired false positive fp_{max} in the bloom model. As the false positive increases with new insertions, it is necessary to look for the item which contains the maximum number of active features n_{max}. The filters having the largest number of active features will achieve larger false positive, yet the upper bound false positive is limited to fp_{max}. According to this discussion, we establish the necessary condition to use a bloom filter model when every filter is built using the same n_{max}, m and k.

In the following sections, we discuss the similarity measures of the bloom model and how this new condition affects to the false positive of these operations, a.k.a. false positive similarity.

5.3.2 Bitwise AND Similarity

The intersection of two bloom filters performs a bitwise AND operation between two filters (Annexe B.2.2) to find out **common insertions in both bloom filters**. Intuitively, one element which is inserted into two different bloom filters will activate the same bits positions in both structures. Figure 5.2 gives an example to represent this fact. As long as one filter represents the active features of one item, the intersection of two bloom filters represents the common active features of two items. Thus, we define a bitwise AND similarity as:

Definition 1 *The AND similarity between two items, A and B, represented by two bloom filters,* $BF_A = \vec{a}$ *and* $BF_B = \vec{b}$*, is given by:*

$$sim(A, B) = card(\vec{a} \cap \vec{b}) = \vec{a} \cdot \vec{b}$$

The low size of filters (regarding real vector representations) and the fast bitwise operations allow to rapidly compare the similarity of two very large item descriptions. This

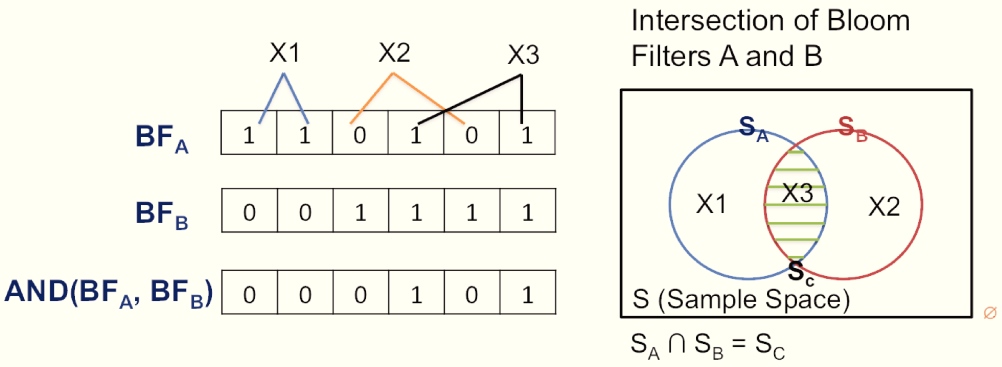


Figure 5.2: AND intersection of two bloom filters. BF_A contains two inserted elements: X1 and X3.BF_B contains two inserted elements: X2 and X3. Thus, the intersected filter contains only one inserted element: X3.

measure might replace the typical vector similarities based on cosine or jaccard measures in recommender systems.

However, filters carry out a false positive ratio to deal with, and hence, the similarity develops a false positive similarity to define, which is indeed the false positive ratio of the resulted bloom filter intersection.

Formally, *the bloom filter probability that a specific bit is set to one in an intersected filter* is defined in Equation 5.1 (see Annexe B.2.2 or [Broder and Mitzenmacher, 2004]).

$$q(bit = 1) = 1 - (1 - \frac{1}{m})^{k \cdot |S_A|} - (1 - \frac{1}{m})^{k \cdot |S_B|} + (1 - \frac{1}{m})^{k \cdot (|S_A| + |S_B| - |S_A \cap S_B|)} \quad (5.1)$$

Although, this probability can be reduced by considering the condition for the bloom model: every filter is built with the same parameters n_{max}, m and k. Filter A represents a set S_A, where $|S_A| = n_A$ is the number of inserted elements. Similarly, filter B represents a set S_B, where $|S_B| = n_B$. Therefore, we know the maximum expected insertions: $n_{max} = n_A = n_B$. Moreover, this limits the maximum cardinality of the intersection, since $|S_A \cap S_B| \leq n_{max}$. As a consequence, the *probability that a specific bit is set to one in both filters* is given by Equation 5.2.

$$q(bit = 1) = \left(1 - (1 - \frac{1}{m})^{k \cdot n_{max}}\right) \quad (5.2)$$

Thus, we define the false positive similarity as the maximum error in the similarity in Equation 5.3, which is the combination of all possible bit states using k hash functions. Note that this error highly depends on k, due to the bit-to-bit comparisons.

$$fp = q(bit = 1)^k = \left(1 - (1 - \frac{1}{m})^{k \cdot n_{max}}\right)^k \tag{5.3}$$

5.3.3 Negating a Bloom Filter

Negating a bloom filter will help us to perform and explain the XNOR similarity measure. Theoretically, the negation of a filter BF_A ($BF_{\overline{A}}$) results in another filter where entries take their complementary value of BF_A. Thus $BF_{\overline{A}}$ might only contain the non-inserted elements of BF_A.

However, due to the k hash functions, insertions may share at most k-bits in the filter BF_A. Therefore, negating a bloom filter has two consequences: (1) inserted elements will appear as non-inserted elements, and (2) non-inserted elements may appear as inserted elements only if all their k indexes are set to 0. Otherwise, non-inserted elements also appear as non-inserted in the negated filter. This fact is represented in Figure 5.3. However, this problem might be minimized by reducing the number of shared bits per element (m/kn). In addition, since the number of bits set to 1 has changed, the false positive of $\overline{A}$ also changes, and it is given by the *probability that a specific bit is zero in the original filter*: $fp = p(bit = 0)^k$ (Equation B.1 in Annexe B.2.2).

5.3.4 Bitwise XNOR Similarity

Common similarity methods measure the intersection of elements to find out mutual similarities. The AND intersection takes common-inserted elements of two sets into account. That is, given two sets, $S_A = \{x1, x3\}$ and $S_B = \{x2, x3\}$, the intersection of both sets is $S_A \cap S_B = \{x3\}$. However, sets S_A and S_B have more in common than this intersection. Actually, the element $x4$ is not in any of these sets, and hence, $x4$ is a common non-inserted element.

As a consequence, we propose to go further in the similarity by considering also the com-

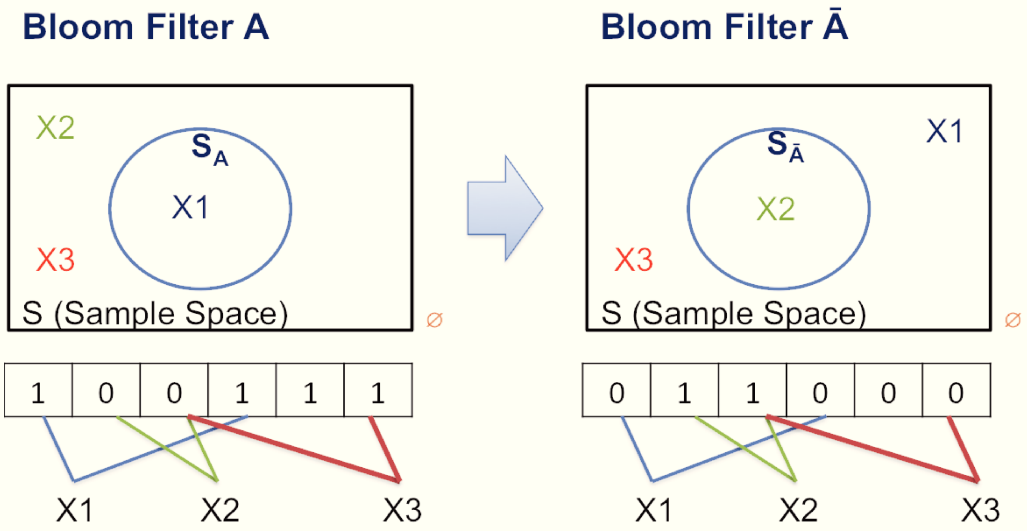


Figure 5.3: Negation of a Bloom Filter (BF). Element X1 is inserted in BF_A. Elements X2 and X3 are not inserted in BF_A. The negation of BF_A, $BF_{\overline{A}}$, contains the element X2 and does not contain the element X1. Yet, X3 is not inserted because it shares a bit with an element which is inserted in BF_A.

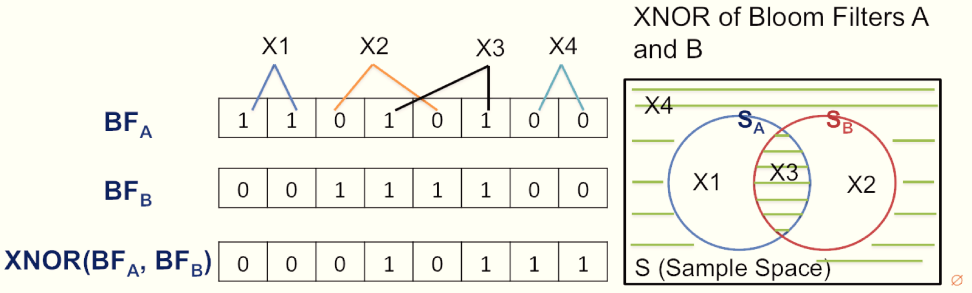


Figure 5.4: XNOR intersection of two bloom filters BF_A and BF_B. BF_A contains two inserted elements: X1 and X3. BF_B contains other two inserted elements: X2 and X3. Thus, the XNOR operation will result in common inserted elements and common non-inserted elements: X3 and X4.

mon non-inserted elements. In fact, this similarity in **common-insertions and common missing-insertions** is the bitwise XNOR operation between two filters. This bit operation sets to one the bits which have similar bit-status. Notice that it is also necessary that every filter is built with the same parameters n_{max}, m and k. Figure 5.4 shows this operation.

Indeed, the XNOR operation between items A and B, represented by two sets, S_A and S_B, is the union of two AND operations: $XNOR(A,B) = S_A \cap S_B \cup S_{\overline{A}} \cap S_{\overline{B}}$. Figure 5.5 graphically shows a set oriented XNOR and its complement XOR operation.

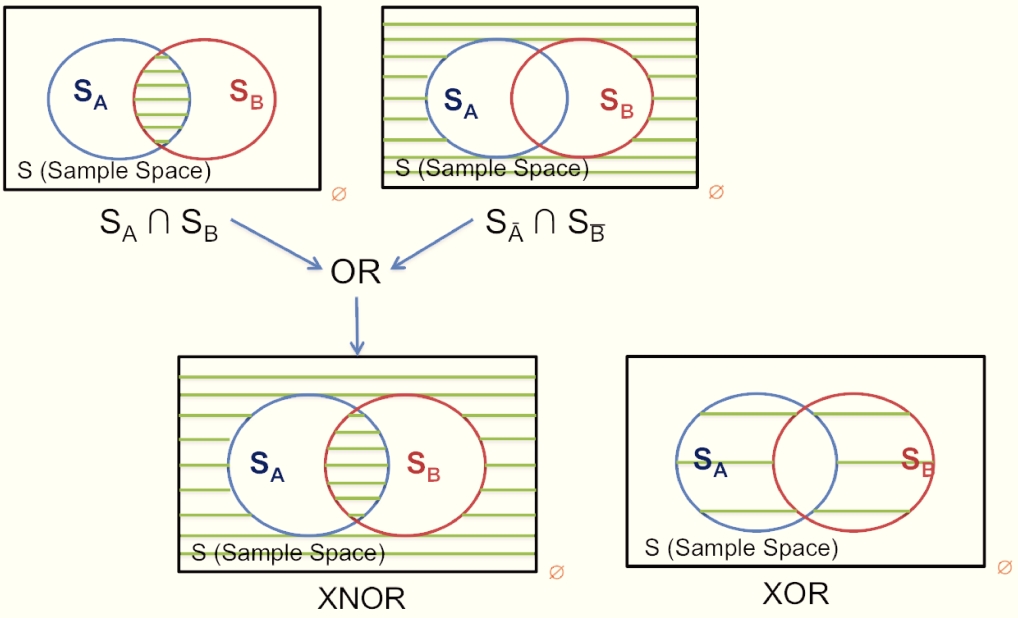


Figure 5.5: Representation of how the XOR and XNOR operations can be extracted from AND and OR operations.

We can observe that XNOR similarity is composed of two AND similarities: common-inserted elements similarity and common-non-inserted elements similarity. Thus, we can compute separately these two AND similarities (explained in the bitwise AND similarity) and merge them. In addition, we propose to weight such similarities by using the weight α.

Definition 2 *The XNOR similarity between two items, A and B, represented by two bloom filters,* $BF_A = \vec{a}$ *and* $BF_B = \vec{b}$*, and with a weight of* α *is given by:*

$$sim(A,B) = \alpha \cdot card(\vec{a} \cap \vec{b}) + (1-\alpha) \cdot card(\vec{\bar{a}} \cap \vec{\bar{b}}) = \alpha \cdot \vec{a} \cdot \vec{b} + (1-\alpha) \cdot \vec{\bar{a}} \cdot \vec{\bar{b}}$$

By varying α, one might avoid the effect of too many common non-insertions in the second part of the similarity. Note that if $\alpha = 1$, this measure becomes the AND bitwise similarity measure explained above.

It is possible to argue that two items may have infinite common non-inserted elements. For instance, a rock and a paper do not have a priori common properties, hence the similar-

ity could be potentially infinite. However, it does not mean they are as similar as an orange and a mandarin can be, and indeed they also have infinitely uncommon features. Thus, a dataset containing rock, paper, orange and mandarin may represent a problem for this measure. Yet, this fact does not happen when the domain of recommendation is fixed and the features to use are defined. In case of multiple recommendation domains, a semantic analysis can be performed in order to limit the features.

Again, we define the false positive similarity as the maximum error in the similarity. For simplicity, let's consider the complementary probability $P(XOR) = 1 - P(XNOR)$ (fact shown in Figure 5.5):

$$P(XOR) = P(BF_A) + P(BF_B) - P(BF_A \cap BF_B)$$

Where $P(BF_A)$ and $P(BF_B)$ are the already known *probabilities that a specific bit is still one*, $q(bit = 1) = 1 - p(bit = 0)$ in filters BF_A and BF_B (explained in Annexe B.2.1. Moreover, $P(BF_A \cap BF_B)$ is the *probability that a specific bit is set to one* in both filters, explained in last Equation 5.1. Taking into account the bloom model condition $n = n_{max}$, one may apply that $q(bit = 1) = P(A) = P(B) = P(A \cap B)$. Hence, the *probability that a specific bit is set to one* in the final resulted vector in the XOR operation is $P(XOR) = q(bit = 1)$. As a result, the *probability that a specific bit is set to one* in the XNOR operation is:

$$P(XNOR) = 1 - q(bit = 1) = \left(1 - \frac{1}{m}\right)^{k \cdot n_{max}} \tag{5.4}$$

Thus, the false positive similarity is given by $P(XNOR)^k$. Operations between identical filters are a singular case. Since all possible elements are represented, this operation results in a totally full vector of 1 being the highest similarity. Hence, membership queries return always true.

5.3.5 Similarity measures and recommender systems

The bloom similarity model does assist the system to deal with high number of items/users features in order to have detailed items/users descriptions. These descriptions are stored

in bloom filters, what allows to reduce the size of descriptions and to perform fast bitwise oriented similarities. The impact on recommender systems are, on the one hand, a lower space complexity and faster similarity operations, and on the other hand, an accurate similarity fidelity. The space complexity and bitwise operations have been already addressed in this section. We would like to develop the second concept of similarity fidelity and how this affects to recommender systems.

Imagine that a recommender system has access to a large number of features to define items and users. The vector representation becomes very large and a feature selection may cause a loss of accuracy in these descriptions. The bloom model allows to make items and users comparisons in very detailed contexts.

On the one hand, the AND similarity measures is fast and accurate enough while comparing very large number of features. Having three items $item_1$, $item_2$ and $item_3$, one can detect that $item_1$ is more similar to $item_3$ than to $item_2$ due to very precise details, such as an ingredient which is in two similar recipes. On the other hand, the XNOR similarity shows comparisons from a different point of view. In this case, one can detect again very particular details that make items similar or different. Another example in this situation is to take into account the similarity about the aspects said in one item/user but not mentioned in others. For example, having three users $user_1$, $user_2$ and $user_3$, one can detect that all users liked drama-comedy movies. However, we know $user_1$ and $user_2$ also like sports, whereas this aspect was never captured for $user_3$. In fact, $user_3$ simply does not like sports and thus interaction with these kind of subjects were never taken. As a consequence, it may be likely that a sport-comedy movie is present to $user_1$ and $user_2$ but not to $user_3$.

We conclude highlighting that these similarities allow to better adapt recommendation to users, since users may be (positively or negatively) aware and sensible to these little details. In this chapter we analyse the accuracy of these similarities regarding other similarities. Other studies to see the impact in real recommendation contexts are part of future work.

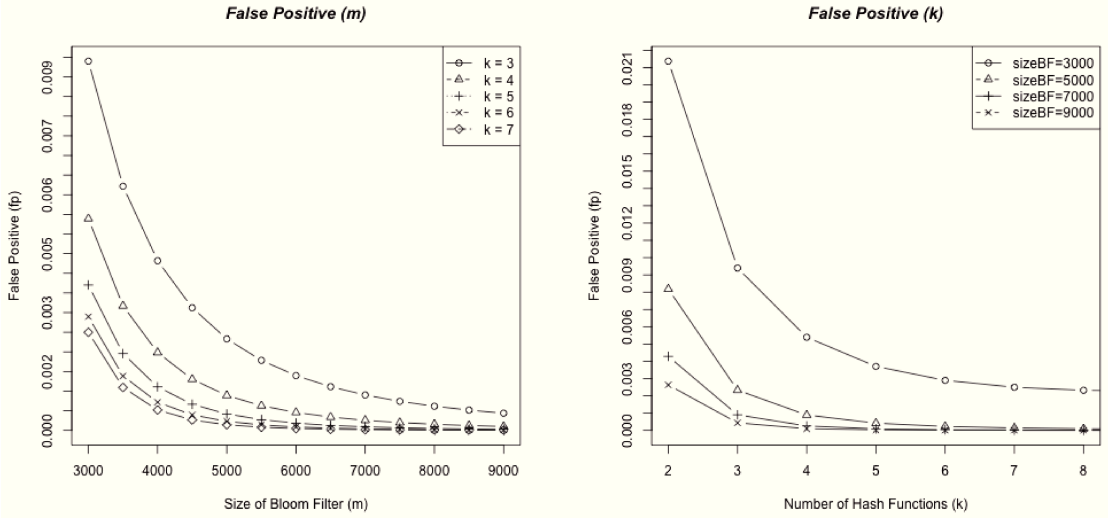
5.3.6 Bloom Model Settings and Trade-Offs

The bloom model clearly depends on the parameters of bloom filters as long as our similarity model relies on the defined AND and OR bitwise operations. It is necessary to analyse the impact of these parameters over the defined bloom similarities. Particularly, we are interested in how these parameters affect the false positive of filters, which is directly related to the false positive similarity between filters, as it has been theoretically demonstrated.

The size m of a bloom filter defines the size of the bit structure, which affects the false positive probability. Larger filter sizes reduce the probability of taking a specific bit $(1/m)$. Therefore, the false positive is reduced as long as the probability of setting the same bit position is reduced as well. Furthermore, the number of hash functions k has also a great impact on the false positive probability. Higher values may reduce rapidly the false positive, but they will quickly fill up the bloom filter. Figures 5.6(a) and 5.6(b) illustrate both cases. In addition, the hash functions particularly affect a bit-to-bit similarity. In fact, the number of shared bits per insertion (m/kn) should be reduced. As long as insertions are associated to k-bits in the bloom filter, higher values in a fixed filter size makes it easier for elements to share some bits positions. Thus, it makes easier to compare a wrong bit state, and hence a true similarity match is more complicated to find.

This m and k trade-off can be seen as a space-time trade-off. Space is represented by the size of the filter. On the contrary, k is the number of hash functions to execute, and hence, higher values entail more time-consumption. Thus, correctly choosing these parameters is crucial in order to have a good performance in a bloom similarity model. However, the choices depend on the requirements of the application and the size of datasets.

In the experimentation phase we compare the performance of different bloom filter and bloom model settings. However, we may conclude that, as long as bit-to-bit operations can be easily compromised by large values of k, we suggested to use larger filters with fewer hash functions to improve similarity matchings.



(a) Fixed k. Increasing m decreases the false positive.

(b) Fixed m. Increasing k decreases the false positive.

Figure 5.6: Evolution of the false positive under different settings k, m. The number of insertions is fixed.

5.3.7 Shortcomings and solutions

Within the sections above, we have explained a very simple case of representing items based on the standard definition of bloom filters, where we have supposed a static dataset. In fact, when new features need to be taken into consideration the number of insertions increases. However, the current bloom filter parameters (size "m" and hash functions "k") can not face such raise and yield to an increase of the false positive. Indeed, these parameters cannot be modified, hence when new features are added to the system all bloom filters need to be re-built according to new parameters. In addition, standard bloom filters do not count the number of insertions of the same element, if necessary. Thus, these filters can not be compared to frequency vectors.

In fact, these situations correspond to static versus dynamic datasets and binary versus frequency vector representations. In our model we use standard bloom filters for simplicity and explanation, yet these drawbacks can be addressed by other bloom filter approaches. On the one hand, dynamic datasets are possible to model by using dynamic/scalable bloom filters [Guo et al., 2010, Almeida et al., 2007]. The main idea is that a scalable bloom filter

is formed by one or more bloom filters. Thus, these filters are built by blocks, and hence by adding new blocks to the filter one may add a new set of features. As a consequence, this avoid to re-build the bloom model when new features are added. In addition, similar probability inductions given in this section can be applied to scalable and dynamic probabilities, and thus our bloom model remains possible in these cases. On the other hand, counting insertions is possible as well by using counting bloom filters [Broder and Mitzenmacher, 2004]. These filters contains new bit-sets to count the insertions performed. Thus, if one insertion has been made several times, filters may approximate the counting. As a result, one may compare counting filters to frequency vectors, with more complexity. Other variances of bloom filters deal with extra compression[Broder and Mitzenmacher, 2004]. These are also interesting filters that can be used for computing items similarity.

In this chapter we consider using standard bloom filters due to the nature of our dataset and experimentations, in order to have an easy comparable dataset and reproducible experimentations. This may be complicated in other dynamic circumstances, e.g. bloom filters in very large datasets using multiple resources, where non public datasets are accessible.

5.4 Experimentation

The experimentations use the dataset given by GroupLens [Cantador et al., 2011] in the domain of movies, which is a merge of the very well known MovieLens and IMDb datasets in the recommendation research field. The dataset is composed of 2113 users, 10197 items, 855598 ratings, 6 features and 104957 possible different values, which are all the available features used to describe our items. In this experimentation we only focus on similarity tasks and we do not take into account users.

Our goal is to compare the performance of the bloom model against two other models namely: a vector model[1] using jaccard/cosine similarities and an order reduction model. The former uses a binary/boolean representation of an item. It takes into account all available features. The latter tries to reduce the size of features by performing a SVD analysis. This experimentation compares four aspects of these models: (1) size, compres-

[1]We use typical vector definition. Highlight that sparse map implementation of vectors may reduce space complexity, yet it increases programming and time complexity in terms of vector similarities.

sion or space complexity, (2) operation time and model time, (3) the similarity fidelity of the AND measure, and (4) the similarity accuracy of the XNOR measure.

This section goes through two kinds of tests. On the one hand, we aim to prove that the filter similarities in bloom model are close to the ones computed by vector similarities (using cosine or jaccard measures) in the vector model. On the other hand, we evaluate the pertinence of bloom filters in recommender systems by comparing the accuracy of several top-K similarities (5, 10, 20, 50, 100, 150, 200, 300 and 500 most similar items to a given one). Notice that a simple top 20 or top 50 is enough in most of recommendation contexts. Then, we compare if our approach reproduces such top-K under the same contexts (similar items and similar features).

The results show that the bloom model reduces the size of vectors up to 97%, keeping a similarity accuracy of 98%. In addition, bloom model outperforms feature reduction methods in terms of similarity fidelity and time performance.

5.4.1 Bloom Filter Representation

This section shows the advantages of using our approach in terms of compression, time-consumption and accuracy.

5.4.1.1 Trade-Off: Compression and Time Analysis

The size of vectors in a vector representation model is given by the total number of features in the database, $N = 104957$. However, the number of active entries in these big vectors is very reduced. The item 3246 has the maximum number of active entries over all items, $n = n_{max} = 237$ (notice that $n_{max} << N$). This shows the maximum information of items in a vector and the worst bloom filter case as well.

As a consequence, to validate the usage of bloom filters against the vector representation model one needs to obtain more reduced filter sizes ($m < N$) and still be able to compute accurate items similarities (low false positive similarities fp). Several configurations are presented in order to compare and find out the best trade-off:

(1) Optimal bloom filters in Table 5.2 computes filter parameters by using the near

Table 5.2: Optimal Bloom Filters.

Bloom Filter	n_{max}	fp	k	m
bloom 0.2	237	0.2	3	794
bloom 0.1	237	0.1	4	1136
bloom 0.01	237	0.01	7	2272
bloom 0.001	237	0.001	10	3408

Table 5.3: Non Optimal Bloom Filters.

Bloom Filter	n_{max}	fp	k	m
bloom M3000K3	237	0.01	3	3000
bloom M3000K4	237	0.005	4	3000
bloom M7000K3	237	0.001	3	7000
bloom M7000K4	237	0.00026	4	7000

optimal false positive value. Indeed, we fix the maximum insertions n_{max} and the desired false positive fp to find the best size m and number of hash functions k. Notice that in the case of using a false positive of 0.001 we obtained 3408 bits, which represents almost 3% of N, thus around 97% of size reduction. This tiny false positive and reduced space is given by the 10 hash functions. On the contrary, this high value may affect our AND or XNOR similarities. In addition, the more hash functions to perform, the slower the system is to build filters.

(2) Non-Optimal bloom filters in Table 5.3 are built by fixing insertions (n), filter size (m) and number of hash functions (k) in order to seek the desired false positive. We vary m and k based on the last optimal computed bloom filter ($m = 3408$ and $k = 10$) to find a good balance in terms of size and number of hash functions for the bit-to-bit operations goal. Notice that for the same false positive 0.001, we obtained a size of 7000 bits, which represents almost 6% of N by only using 3 hash functions. This setting is a priori the most interesting for our comparison purposes

Table 5.4 shows interesting comparisons in terms of space and time-consumption. The machine used is a MacOS 4Go RAM with 2 cores (2.53GHz). Three models are compared: Vector Model (VM), Optimal Bloom Model (OPM) and Non-Optimal Bloom Model (NOPM). To build vectors and filters, a database access was required. Query time was around 200 ms (not included in these results). One may observe that building bloom model takes extra time due to the hash functions, however, this is acceptable as there is a high space reduction. In addition, operations among bloom models can perform faster due to two facts: (1) bitwise operation are faster than jaccard/cosine similarities, and (2) bitsets are smaller.

Moreover, it might be interesting to reconstruct a vector from a bloom filter to compare

Table 5.4: Vector and Bloom Models trade-offs.

Item Representation	VM	OPM	NOPM
Size (bits)	104957	3408	7000
Hash Functions	-	10	3
Building model (sec)	6.58	8.94	8.32
AND Similarity (ms)	-	0.001	0.003
XNOR Similarity (ms)	-	0.004	0.006
Reconstruction (ms)	-	127.28	141.27
Jaccard Similarity (ms)	0.4	2.6	2.3

vector and bloom models under the same conditions and similarities. Reconstruction time is given by the features loading (140 ms) and the features hashing (145 ms), which are performed only once. Then, a single vector reconstruction is created by querying filters. This is also shown in Table 5.4.

5.4.1.2 Singular Value Decomposition (SVD) model

As explained in Chapter 4, the SVD is a dimension reduction technique. Indeed, it is a factorization model that decomposes a big matrix in three smaller matrices (left-eigenvectors, eigenvalues and right eigenvectors) in such a way that the multiplication of the three is an approximation of the original matrix. The eigenvalues shows which are the principal axes to consider, and thus the rank of the matrix may be reduced.

In the field of recommender systems this model helps to focus on particular attributes and features to represent items/users instead of using large and sparse representations. The main goal is to focus only on the important axes of the model which well represents items/users. However, this model losses some precision while neglecting features in order to gain in space. We applied SVD to the set of items-features (a matrix of 10197 items and 104957 features) to find an accurate reduced matrix representation and to compare its size and similarity fidelity against the vector representation model and our bloom model.

The threshold we impose to SVD is the number of eigenvalues and eigenvectors to seek. We only consider the bit-size of eigenvalues for simplicity in size and compressions comparisons to the bloom filter representations. Each eigenvalue has a double bit precision by using a 32 bits representation. Thus the total rank representation may use 9600 bits,

which is already larger than the 7000 bits of our biggest bloom filter. This makes that the number of the number of non-zero eigenvalues to find should not overpass 300.

The comparisons were stopped because the item similarities that one may achieve by using the reduced SVD model were not accurate. Indeed, the items-features matrices are so sparse that the SVD can not find a very low rank model. This may be solved by increasing the rank of the matrix, yet the bloom model have already demonstrated very good similarity fidelity in a much more reduced item space representation. In addition, the large dataset made the SVD model very memory and timely expensive.

5.4.2 Fidelity of the AND similarity

Our bloom model offers two relevant performances: reduced size and similarity fidelity. In this section, we evaluate the accuracy of the AND Similarity measure.

We compute the similarity between filters, and thus items, by performing the two-by-two bitwise AND operation. First, the bit operation returns the intersected filter between two filters. There are two ways to exploit such results: the *cardinality representation* and the *set representation* of the bloom filter. The former compares two filters by using their bit-set vectors. It is fast but it highly depends on low values of k. The latter compares two filters by their set representations. In fact, it reconstructs the AND intersected bloom filter into a N dimensional vector by querying it. Hence, it is slower, yet it allows to compare bloom model and vector model similarities under the same conditions, since items in both models are represented as vectors of dimension N.

On the one hand, we reconstruct the set representation of the resulted AND operation. The first test aims to check whether the bloom model is loyal to the similarity degree of items, i.e. similarity fidelity. Hence, we compare item similarities in both models: vector similarity model (using jaccard similarity) and bloom similarity model (using AND bitwise similarity). We focus on item 3426 which has more active features, and thus the bloom filter with the most insertions and highest false positive ratio. This is our worst case comparison, since it is more likely to have bits conflicts in a filter comparison. Figure 5.7 shows the degree of similarity of this item to other items in the dataset for both approaches. It demonstrates the high fidelity in the similarity of the bloom model (case bloom 0.001 in

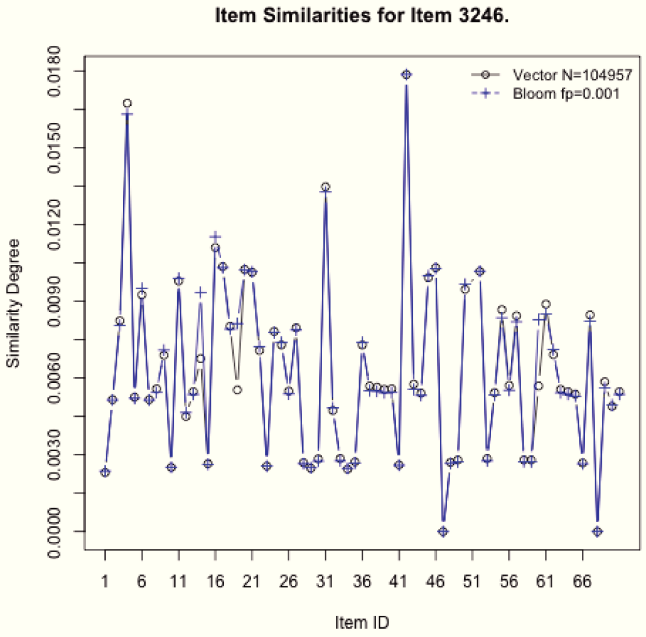
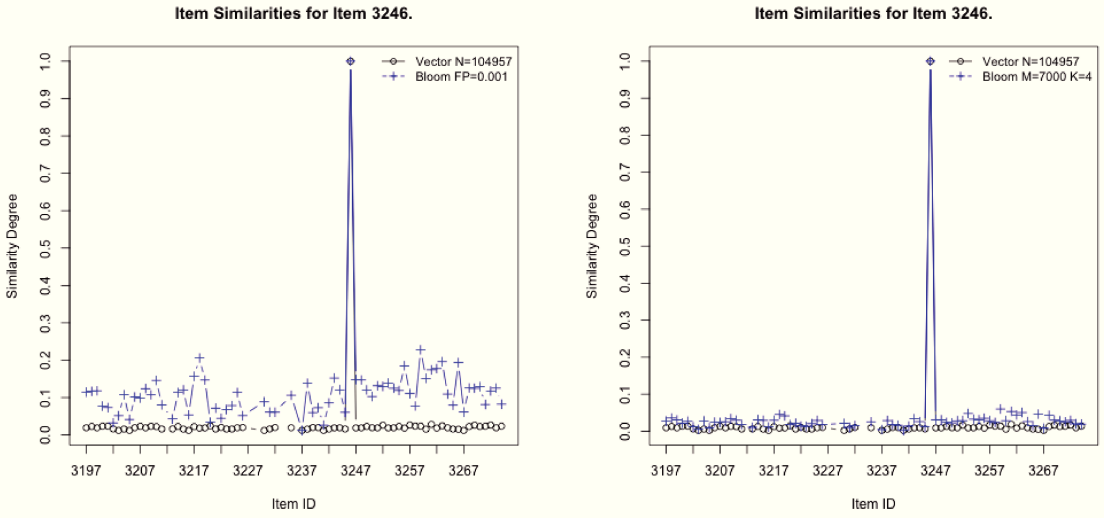


Figure 5.7: Degree of Similarity of the item 3246 with several items. Bloom Filter (n=237, fp=0.001) has been reconstructed into a vector of size N.

Table 5.2) while reproducing the vector model jaccard similarity.

On the other hand, one can use the cardinality of the resulted bloom filter intersection. Thus, the number of hash functions k highly affects this similarity due to the shared bits of insertions. Figure 5.8(a) shows an optimal bloom filter case where the high value of k causes the points sparsity. In fact, high values of k associate inserted elements to more number of bits. Thus, it makes difficult to perform comparisons based on the cardinality of the intersection. Increasing m and decreasing k the bloom filter fills up slowly with insertions, and thus better cardinality comparisons are possible as shown in Figures 5.8(b) and 5.9(a). Finally, in Figure 5.9(b) we made a zoom over few values in the last and best configuration to better appreciate these values. Very little differences are observable in terms of similarity fidelity, whereas as the size of the filter is almost 6% of N ($N = 104957$).

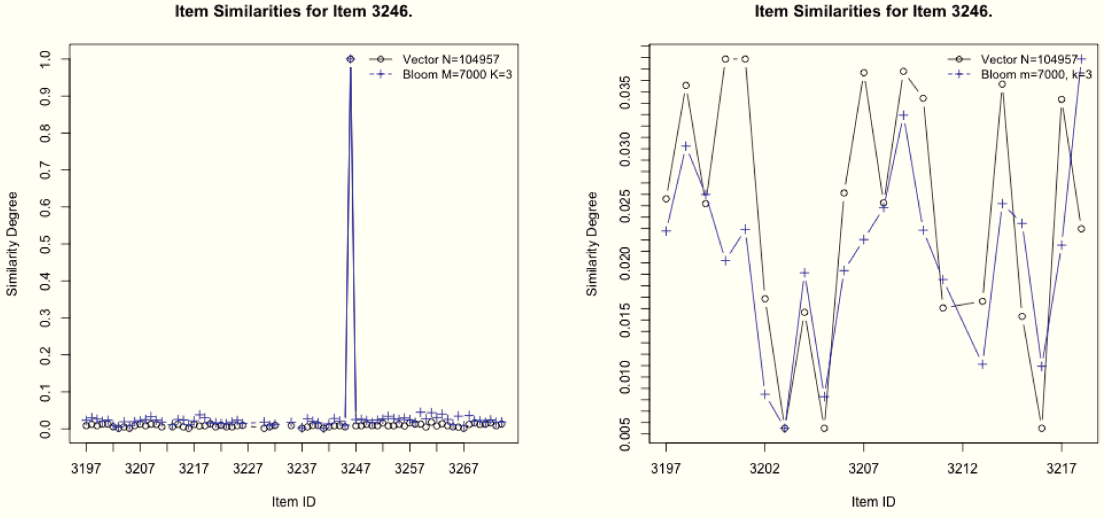
However, similarity models can be loyal while reproducing such item similarities, yet tiny differences may have big differences in a top-K comparisons. We aim to compare the ability of our bloom model to find out the K-most similar items for the current item



(a) Bloom Filter (m=3408, k=10).

(b) Bloom Filter (m=7000, k=4).

Figure 5.8: Trade-Offs for a Bloom Similarity Model.



(a) Similarity fidelity.

(b) Similarity fidelity. Zoom over 5.9(a).

Figure 5.9: Bloom Filter (m=7000, k=3). Trade-Off for a Bloom Similarity Model.

3246. Again, this is our worst case. Thus, we request the vector similarity model to return several top-5, 10, 20, 50, 100, 150, 200, 300 and 500 most similar items by using the jaccard similarity. The goal of the bloom filter similarity model is to reproduce such

tops: the same items should appear in both tops (in this test the order of items is not taken into account). As a result, by comparing tops, one knows the correct presence of items (True Positive (TP)), the correct absence of items (True Negative (TN)), and the errors (False Negative (FN) and False Positive (FP)). As a consequence, the accuracy of the system is defined by Equation 5.5:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.5}$$

The values for this accuracy vary between 0 and 1, where the value of 1 is the highest possible accuracy.

The most important parameters of our bloom model for similarity comparisons purposes is the false positive in bloom filters because it may derives in false positive similarities. Thus, in order to compare both similarity models under the same conditions, we reconstruct the bloom filter representations from Table 5.2 into vector representations of size N. The false positive ratio will make the difference between a real vector and a reconstructed vector, and hence, comparisons in similarities are possible to perform.

Figure 5.10 shows the results for these tops comparisons. Notice that bloom filter representations may achieve almost a perfect accuracy fidelity in very reduced sizes. In fact, the bloom filter with a false positive of 0.001 and 3408 bits achieves an almost perfect score in these presented top similarities.

5.4.3 The XNOR similarity

The XNOR Similarity shows common insertions in two bloom filters, but it also shows the features that likely have not been inserted in any of the bloom filters, i.e. common insertions and common non-insertions. In this test we compare whether the bloom model correctly find similar items by using this XNOR operation. Again, the false positive ratio of bloom filters has impact over the filter similarities. We consider the settings of Table 5.2 to compare our performances.

We compute the XNOR operation in vector models, and we compare this to the XNOR operation in the bloom filter similarity model. Once the bloom filter operations have been

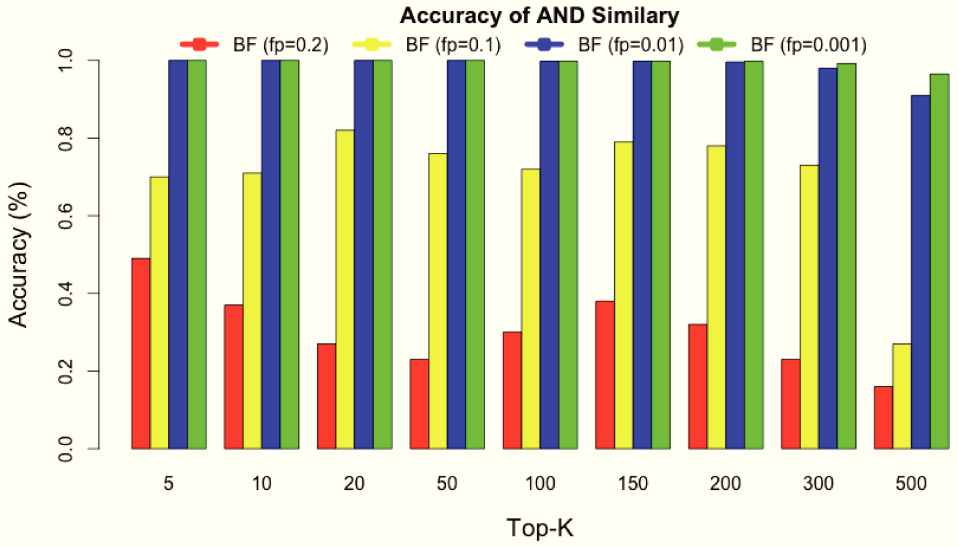


Figure 5.10: Accuracy of the Bloom Filter Similarity Model against the Vector Similarity Model. Bloom Filters with fewer false positive values achieve better accuracy in tops comparisons.

performed, we reconstruct the resulting filters into vectors of N dimensions for simple comparisons to vector models. The XNOR similarity equally balances the similarity of common insertions and common non-inserted elements. That is, $\alpha = 0.5$. However, one may change the weight in this similarity by giving more relevancy to one aspect or another, e.g. giving more relevancy to common inserted elements versus common non-inserted elements. This is traduced in a tendency to AND similarities or complementary AND similarities.

The similarity performance evaluation is again made in terms of accuracy. We compare both XNOR top-K looking for the presence and the absence of items among the top-K. The results of this similarity is shown in Figure 5.11. One may see how the bloom model correctly finds the same similarities given by the vector model in much fewer sizes. Especially, very low false positive configurations into account larger number of features in the similarity, whereas the space complexity is again much more reduced.

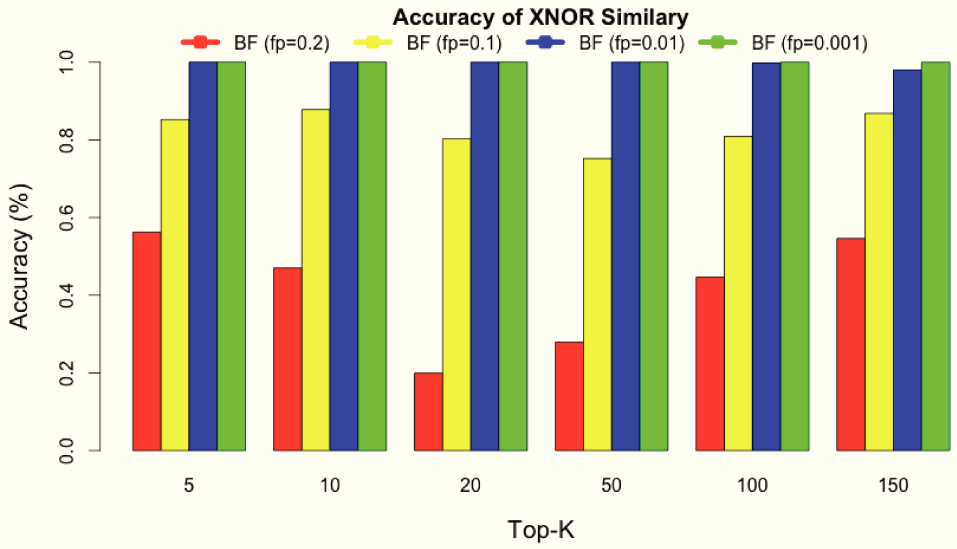


Figure 5.11: Accuracy of the Bloom Filter Similarity Model against the Vector Similarity Model. This similarity uses the XNOR operation.

5.5 Discussion

Recommender Systems, particularly content-based and hybrid recommendation techniques, usually represent items/users by a large set of features. This allows to make correlations among data and to compute similarities among items/users. Then, they are used to achieve and adapt the recommendation of items to users.

The accuracy of the similarity depends on the selected features. Generally, the more features are used the better the similarity might be. In addition, items' features are typically represented by vectors, which are large and sparse. Hence, increasing vectors' sizes makes similarity computations slower. Some authors have suggested to use dimensionality reduction techniques to capture only the most relevant features and keep a lower space complexity. This approach, on the contrary, leads into a loss of accuracy in similarity comparisons.

The main motivation behind our contribution is that large items/users description add space complexity and time consumption, although they have a great similarity detail. We propose a new method for representing the items/users based on bloom filters. Our main

goal is to use bloom filters in order to alleviate these constraints and to create similarity measures based on bloom filters. As a result, we focused on three aspects: (1) the size compression of bloom filters in a recommender system context, (2) the usage of AND operations as a similarity measure in bloom filters to consider common insertions, and (3) the usage of XNOR operations as a similarity measure that takes into account not only common inserted items, but also common non-inserted items.

The experimentations performed on a public dataset [Cantador et al., 2011] show that the bloom filter representation highly reduces the size of vector representations (94-97% per vector) while keeping a high fidelity in the item similarity (accuracy of 98%) in comparison with standard approaches.

This chapter has faced the items' and users' large vector representation challenge. Our approach allows to reduce memory consumption in recommendation process as long as to use very detailed items' and users' descriptions. The next chapter (Chapter 6) focuses on the cold-start challenge in recommender systems. Particularly, we focus on collaborative filtering techniques only, which do not exploit the items' and/or users' descriptions.

Chapter 6

Active Learning to Cope with New User Cold-Start

"A man must be big enough to admit his mistakes, smart enough to profit from them, and strong enough to correct them."

- John C. Maxwell

One paper derived from this chapter has been submitted to ICDM 2016: *"Enhancing New User Cold-Start based on Decision Trees Active Learning by Using Past Warm-Users Predictions"*; Manuel Pozo, Raja Chiky, Elisabeth Métais and Farid Meziane. Acceptance Notification on September 9th.

6.1 Motivation

Recommender Systems have demonstrated a great accuracy while predicting the interests of "warm-users", i.e. users whose some interests are known. However, they still suffer from cold-start problems, such as the new user cold-start and new item cold-start, a.k.a. "cold-user" and "cold-item" respectively.

The cold-start is the situation in which the recommender system has no or not enough information about the (new) users/items, i.e. their ratings/feedback; hence, the recommendation to users (or of items) are not well performed. On the one hand, the item cold-start can be alleviated by using the item's attributes in content-based and hybrid recommendation techniques. Moreover, this information is easily available. On the contrary, the user cold start is more difficult to deal with since the new user needs to deliberately present

her attributes (e.g. age, genre, studies, etc.) and/or expresses her interests in items (i.e. ratings/feedback). However, users are not willing to give much information and evaluate many items [Rubens et al., 2011, Elahi et al., 2014].

This issue is commonly encountered in collaborative filtering recommendations as they rely mainly on the users' feedback to predict future users' interests [Su and Khoshgoftaar, 2009]. Moreover, the recommendations' accuracy is directly related to the users' satisfaction and fidelity [Rubens et al., 2011]. New users start evaluating the system from their first usage and this makes the recommendation process a challenge for both academia and industry [Golbandi et al., 2011]. Users may do not trust the recommendations given and may leave before the system learns to return proper recommendations.

The current techniques to cope with the new users cold-start are categorized into passive learning and active learning:

- Passive collaborative filtering techniques learn from sporadic users' ratings; hence learning new users preferences is slow [Karimi et al., 2015a]. Other techniques propose correlations between users and/or items by using the users/items attributes [Kantor et al., 2011], such as content-based [Peis et al., 2008] and hybrid methods [Karim, 2014]. Many of these techniques were discussed in the Chapter 2. However, dealing with such features slows down the process and adds complexity and domain dependency.

- Active techniques interact with the new users in order to retrieve a bunch of ratings that allows to learn the users' preferences. A naive but extended approach is to question users about their interests and get their answers [Rashid et al., 2002]. Such questions may include: 'Do you like this movie?', with possible answers such as: 'Yes, I do'; 'No, I do not'; 'I have not seen it'. In fact, this process can be applied for cold-users in a sign-up process (a.k.a. Standard Interaction Model) or for warm-users (a.k.a. Conversational and Collaborative Model) where users can provided new preferences to the system; hence the system can better learn all users preferences [Rubens et al., 2011].

Figure 6.1 represents a functional diagram for both passive and active learning ap-

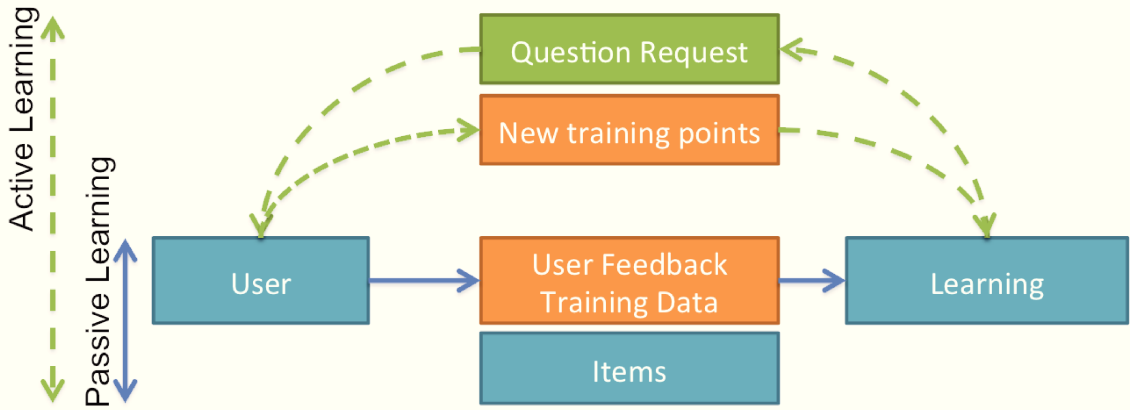


Figure 6.1: Passive and Active Learning pipelines [Rubens et al., 2011].

proaches. Passive learning algorithms follow a straight pipeline in which the techniques have to wait for new feedback. This may slightly adapt the recommendations but this process requires several "recommendations trials" until starting to learn the real users preferences. Active learning perform a backward pipeline to involve the user in the learning process by actively demanding for new ratings or feedback.

In this chapter we focus on active learning to cope with the new user cold-start. Particularly, we study the collaborative filtering based techniques because (1) they have already demonstrated a great accuracy for warm users recommendations, (2) they allow a fast generation of personalized recommendations to new users, and (3) they only require users' ratings. In fact, it has been demonstrated that few ratings from (new) users are more informative than using users' attributes making it possible to obtain more accurate recommendations in cold-start situations [Pilászy and Tikk, 2009]. As a consequence, we face a very identified problem held to the advantages and drawbacks of the proposed techniques.

Active learning techniques propose a set of questions and answers to users, a.k.a. the questionnaire. In our context, the questions are items and the answers are the users' preferences to these items. The questionnaires are batch-oriented (several questions at once) or sequential (the questions are presented one after the other). In addition, the questions can be non-personalized (the same questions are asked to all users, e.g. most sold items), or personalized (the next questions depend on the users' past answers). However,

users are not willing to answer many questions [Rubens et al., 2011, Elahi et al., 2014]. Therefore, the main challenge in active learning is to present short but very informative questionnaires (a maximum of 5-7 questions [Golbandi et al., 2011]). This maximizes the information retrieved from users and minimizes users' efforts [Harpale and Yang, 2008].

The personalization of the questionnaires lead to a progressive understanding of the user's preferences. In fact, the personalization of the questionnaires is close to a recommender system concept, although the latter seeks the items the user likes and the former seeks the items the user recognizes. In this context, we specifically focus on an optimization of the prediction accuracy, since it is directly linked to users' satisfaction [Rubens et al., 2011].

Current techniques for personalizing questionnaires in active learning collaborative filtering are based on decision trees. These techniques analyze the available warm users' ratings in order to find out which items to propose to users. However, their effectiveness in small datasets has not been probed. Moreover, we believe that taking into account warm users' predictions may enhance these techniques. In this chapter we suggest to exploit both available warm users' ratings and warm users' ratings predictions in order to improve the questionnaire. The experimentation shows that our approach enhances previously suggested ones in terms of accuracy and in using a smaller number of questions.

The remaining structure of this chapter is organized as follows: Section 6.2.2 presents the state of the art for active learning using decision trees techniques. Section 6.3 gives the background and notation used for decision trees. Section 6.4 presents our contribution to enhance active learning based on past warm users' rating predictions. Section 6.5 shows the experimentations performed and the results of our approach. Finally we conclude and present our future works in Section 6.6.

6.2 Related Work in Active Learning

Active learning is a data acquisition method that not only helps the system to learn cold users preferences, but it lets warm users to clarify and better express their preferences as well. Thus, the user is more self-conscious about her own preferences while the system

Figure 6.2: Example of candidate items and close-form show to users.

continuously presents and discover new items for the user. This represents a large exploration/exploitation challenge in recommender systems [Kantor et al., 2011] and we will not discuss it in this chapter.

Active learning proposes the (new) user to fill-out a questionnaire that will retrieve the first user's preferences. The questions are simple and possible answers are limited. For instance a simple answer questionnaire may be: 'Do you like this movie?': 'Yes, I do like'; 'No, I do not like'; 'I have not seen it'. Also, a 'Rating Stars'-Unknown questionnaire might look like: 'Can you rate this book?': 'I hate it (1 star)'; 'I do not like it (2 stars)'; 'It is acceptable (3 stars)'; 'I like it (4 stars)'; 'I love it (5 stars)'; 'I have not read it'. Figure 6.2 represents a similar example.

As it can be noticed, the type of questions and answers represent a cost for the user. Therefore, there is a qualitative-quantitative trade-off in questionnaires: short number of questions and short number of answer's choices lead into a quick full-filling of questionnaires, but the information retrieved could be not enough to understand new users' preferences, whereas large questionnaires with multiple answer's choices may challenge users' willingness to full-fill (e.g. boring, too much rating debate, etc.). The personalization of the active learning aims to reduces the users' effort and to maximize the system's learning process for the effectiveness of the questionnaire [Zhou et al., 2011].

In this research, we give a global related work of active learning, and we focus on active learning techniques in the domain of collaborative filtering recommendations, particularly those using decision trees because: (1) the sequential question paradigm allows a good personalization of the questionnaire [Elahi et al., 2014], and (2) they aim to well profile

a new user by posing as less questions as possible [Golbandi et al., 2011].

6.2.1 Personalization of questionnaires and strategies in active learning

On the one hand, the personalization of the questionnaire increases the information retrieved from users. Randomly selected candidate items (questions) are possibly not recognized by the new users. The static non personalized questionnaires show always the same candidate items regardless the user (eg. most sold items), and hence the evolution of her tastes is not well captured. Personalized questionnaires alleviate these drawbacks. On the contrary, they may carry out a waiting time between questions [Karimi et al., 2011a], and the users are not willing to wait. Thus, the ideal questionnaire may intelligently present personalized candidate items and be fast to react to answers.

On the other hand, the active learning techniques have one or many strategies to pick up adequate candidate items. Some of them are given below, for a further detailed classification of these criterion strategies please refer to [Rashid et al., 2002, Elahi et al., 2014]:

- Popularity, Variance and Coverage. Most popular items tend to have higher number of ratings, and thus they are more recognized. Popularity-based questionnaires increase the "ratability"[Carenini et al., 2003] of candidate items in order to obtain more number of feedback, although very particular interests of new user preferences, out of popular items, are not captured. In addition, items with low rating variances are less informative. Thus, variance-based questionnaires show the uncertainty of the system about the prediction of an item [Boutilier et al., 2002]. On the other hand, the item's coverage (i.e. number of users related to this item) can lead to create interesting rating's correlation patterns between users.

- Entropy. This strategy uses information theories, such as the Shannon's Theory [Shannon, 2001], to measure the dispersion of items ratings and hence to evaluate items informativeness. This technique tends to select rarely known items. In addition, entropy and popularity are correlated, and they are very influenced by the users' ratability (capacity of users to know/rate the proposed items) [Rashid et al., 2002].

- Optimization. The system selects the items from whose new feedbacks may improve a prediction error rate, such as MAE or RMSE. Indeed, this is a very important aspect in recommender systems since error reduction is directly related to users' satisfaction [Rubens et al., 2011]. Other strategies may focus on the influence of queried item evaluations (influence based [Rubens and Sugiyama, 2007]), the user partitioning generated by these evaluations (user clustering [Rashid et al., 2008], decision trees [Golbandi et al., 2010]) or simply analyse the impact of the given rating for future predictions (impact analysis [Mello et al., 2010]).

Three very well known non-personalized and batch-oriented strategies are: (1) Entropy0 relaxes the entropy constraints by supposing that unknown ratings are ratings equal to '0' (changing a 1-5 rating scale to 0-5 rating scale), hence a high frequency of '0' tends to decrease the entropy, (2) "LPE" (Logarithmic Popularity Entropy) chooses candidate items regarding their popularity and rating entropy [Rashid et al., 2002], and (3) "HELF" (Harmonic Entropy Logarithmic Frequency) balances the entropy of candidate items against the frequency of rating repetitions [Rashid et al., 2008]. However, the need of personalization in questionnaires has changed batch-oriented into sequential-oriented based questionnaires. This evolution is shown in the works: [Karimi et al., 2011a] [Karimi et al., 2011b] [Karimi et al., 2011c] [Karimi et al., 2012] [Karimi, 2014]. Within these papers, the authors have explained the importance of rapid online questionnaires and the ratability factor over candidate items in order to capture the users preferences. In the next section we focus on personalized questionnaires only.

6.2.2 Active Learning for Collaborative Filtering

The first appearance of active learning for new users cold-start was in [Kohrs and Merialdo, 2001], although the first step for creating sequential personalized questionnaires was suggested in [Harpale and Yang, 2008]. In this paper, the authors enhance a similar approach in [Jin and Si, 2004] by assuming that users may do not be able to rate presented items and thus relaxing initial assumptions. They suggest a probabilistic collaborative filtering that uses bayesian networks to learn the candidate items entropy. The candidate items are presented sequentially and the whole model is re-adjusted according to past

answers. However, these models do not perform well in on-line questionnaires because questions' answers lead into a time-consuming model update. In [Karimi et al., 2011a] this idea was extended by questioning only about the most popular items. This reduces the number of items to focus on and results in faster models. In [Karimi et al., 2011b] they applied active learning for matrix factorization. They believe that solving the new user cold-start is an optimization problem, which explores the latent space given by the matrix factorization to get new users parameters, then it exploits and adjusts these users parameters. Recomputing the whole matrix with each new rating is not tractable, and hence they propose a fast-online updating [Rendle and Schmidt-Thieme, 2008] after each answer.

One technique that is very meaningful in active learning is user partitioning. It allows to group users of similar tastes into clusters or nodes, and then tries to find out to which group the new user belongs to. In [Rashid et al., 2008] the authors assumed that finding the correct users neighbors will improve the information gain of the questions presented to users. They presented the Information Gain through Clustered Neighbors (IGCN) algorithm to adjust the entropy of the items by taking into account only those users who match with the new user's profile.

In [Golbandi et al., 2011] the authors use non supervised ternary decision trees to model the questionnaire. The decision trees are built off-line to be completely available for new users that receive the questions sequentially. To move to a new question they answer the current one by clicking on one of the three possible answers ('like', 'hate' and 'unknown'). The users' answers lead to a different child node of the ternary decision trees. This creates a personalized tree path that depends on the past users' answers. On the other hand, this technique uses a collaborative filtering approach to build the decision trees. Using available users' ratings, they seek the best discriminative item in order to split the population of users into three nodes (users who liked, those who hated and those who do not know this item). The best item is the one which minimizes a statistical error within the users' ratings of the node. In order to evaluate the performance of the tree, the authors add labels to the candidate items by using the item average prediction method and demonstrated that their technique improves the accuracy of the system.

Recent literature seems to focus on decision trees to handle questionnaires and on

matrix factorization as a prediction model. The authors in [Zhou et al., 2011] suggest to integrate decision trees construction into the matrix factorization model. They call it "Functional Matrix Factorization". The first step is to learn about the underlying item feature vectors. In the second step, the users are defined as ratings' vectors and one function maps these users into an underlying users features vector. This function is responsible for creating a ternary decision tree ('like', 'dislike', 'unknown'). Hence, matrix factorization is applied to learn both function and items-features through an iterative alternating minimization process, which is performed in each node of the three. In [Karimi et al., 2014], the authors claim that [Zhou et al., 2011] is computationally too expensive. On the contrary, they suggested incorporating the matrix factorization into decision trees. They first build the decision tree as in [Golbandi et al., 2011]. Each node represents an item with an associated rating prediction label. The goal of the matrix factorization is to improve this label. Indeed, they train one matrix factorization model for each level of the tree, and aggregate users within the nodes into a pseudo-user for whom a prediction about the candidate item in the node is performed. This new predicted label replaces the last one.

Very recently, in [Karimi et al., 2015b] the authors assumes that warm users can be thought as new users from whom some ratings are known. Thus, this is seen as a supervised decision trees which internally reduces the accuracy of the technique by picking the best discriminative items. Moreover, they split the tree nodes into six, a 1-5 natural scale rating and an unknown node. This technique improves [Golbandi et al., 2011], and is later enhanced by (1) taking into account the most popular items only [Karimi et al., 2011a], and (2) using matrix factorization to improve the prediction labels assigned to the tree nodes [Karimi et al., 2014].

The approaches in [Golbandi et al., 2011, Karimi et al., 2015b] use the big but not available Netflix dataset [Narayanan and Shmatikov, 2008]. However, these approaches are not tested on smaller datasets, especially in terms of the number of users/ratings. In fact, a low number of users and ratings may produce an early pruning of some brunches of the tree. Moreover, early nodes with less data reduce the accuracy of the item's average method and the discriminative items choice.

6.3 Background and Notation

The goal of decision trees in active learning is to split the users' population in groups of users' preferences. Thus, the users within the same tree's node tend to share similar preferences. By going deeper inside the tree, these groups are refined and preferences are better detected. These techniques in a collaborative filtering context have only access to the available users/items feedback, i.e. ratings. As a consequence, they aim to find out the discriminative items that, in each node of the tree, efficiently split the users' population depending on the users' feedback to these items.

This idea is very useful to cope with the new users cold start. One node is represented by one question about the discriminative item, e.g. 'Do you like this movie?'. The answers are the possible feedback of the new user to this item, e.g. 'like', 'dislike' and 'unknown'. Every answer leads to a different question. Therefore, when new users fill-out the questionnaire, they are indeed following a personalized path which tries to detect to which group of users' preferences they match the best.

In this chapter we will use the following notation for active learning using decision trees algorithms. Formally, let R be the available ratings. The rating of a user u in an item i is defined by $r_{u,i} \in R$. In addition, let t be a node in the decision trees. We define U_t, I_t, and R_t as the set of (warm) users, items and ratings currently in the node t. Furthermore, $R_t(u)$ and $R_t(i)$ are ratings of the user u and item i in the node t. In addition, our main contribution exploits the predictions over the existing R. Thus, we define P as the predicted set of R, so that for each $r_{u,i} \in R$ there is a prediction $p_{u,i} \in P$. The set P is computed by using collaborative filtering techniques, e.g matrix factorization. Highlight that the number of users, items, and entries in R and P are the same. Particularly, we do not aim to predict sparse values or not known ratings from all possibles entry combinations of U_t, I_t. Finally, P_t is the set of predictions currently in the node t, and $P_t(u)$ and $P_t(i)$ are the set of users and items predictions in the node t. Table 6.1 resumes these notations.

Table 6.1: Notation used in decision trees.

Notation	Description
R	Set of ratings
P	Set of predictions
u	User
i	Item
$r_{u,i}$	Rating of user u in item i
$p_{u,i}$	Predicted rating of user u in item i
t	Current node of the tree
R_t	Set of ratings in node t
P_t	Set of predictions in node t
U_t	Set of users in node t
I_t	Set of items in node t
$R_t(u)$	Set of user's ratings in node t
$R_t(i)$	Set of item's ratings in node t
$P_t(u)$	Set of user's predictions in node t
$P_t(i)$	Set of item's predictions in node t

6.4 Active Learning Decision Trees

This section introduces our contribution. We suggest exploiting not only the available ratings from warm users, but also the predictions made by collaborative filtering algorithms over these available ratings. We first give some considerations to decision trees for small datasets as long as the current state of the art has not been tested in these environments. Second, we illustrate our core idea using a real use-case for a better discussion and understanding. We use two datasets from MovieLens [1]. Besides, we show the properties of the Netflix dataset [Narayanan and Shmatikov, 2008] for datasets' size comparisons. These datasets are later used in Section 6.5 to evaluate our approach and their properties are given in Table 6.3. Third, we present the techniques and algorithms for decision trees in non supervised and supervised techniques. In addition, a complexity analysis of our algorithms is provided.

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

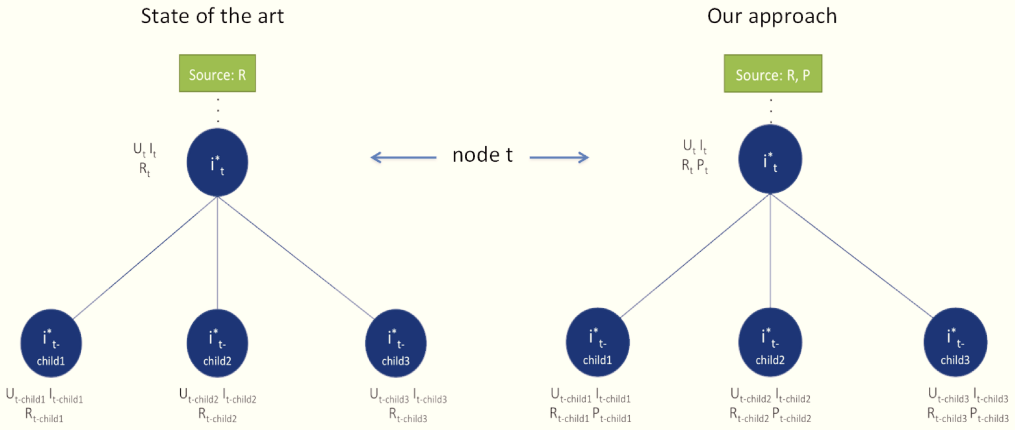


Figure 6.3: Illustration of our approach and comparison to the state of the art.

6.4.1 Decision Trees in small datasets

The active learning in collaborative filtering is more challenging in small datasets since there is less data make correlations with. Particularly, we discuss the concentration of ratings and users in the nodes of the tree in three co-related aspects: (1) the number of child nodes in a tree, (2) the number of ratings in a node, and (3) the number of users in a node. We do not strictly address this, although as it will be shown in the experimentation phase, we improve the current approaches under this context as well.

The number of child nodes in which the current node t is split plays an important role in small datasets. In big datasets, using a large scale of answers may yield in a better accuracy while classifying users' preferences. The large number of users in the dataset is split among all child nodes. Moreover, the less informative node, the group for 'unknown' is always the most populated due to the sparnse nature of datasets [Golbandi et al., 2011]. As a consequence, the tree child nodes containing small number of users tend to be pruned faster. In addition, less populated nodes may result in an unstable performance since there is not much data to correlate. In datasets with less number of users we suggest using short scale answers, such as 'like', 'dislike' and 'unknown'. This allows to aggregate users into the same node to ensure the usefulness of the tree.

Moreover, the number of ratings in a tree node helps the system to analyse current

items, to find out the best discriminative items and to assign a prediction label to them. This factor is used in [Golbandi et al., 2011] as a stopping criteria. The authors claim that there is a very little gain in going further in analyzing nodes with few ratings. Nevertheless, we consider that the number of users should be taken into considerations as well. The number of ratings in a node is directly related to the number of users in this node. This is important in the field of recommender systems since users' preferences with higher number of ratings are better detected. On the contrary, having less number of ratings per user may lead to a bad understanding of users' preferences. In decision trees, nodes containing few users but many ratings perform better in preference detection than nodes containing many users with fewer ratings. As a result, taking into account ratings only may yield to unstable performances in small datasets.

6.4.2 Warm users predictions in decision trees

Users are not willing to rate too many items. Hence, the main challenge of active learning is to learn the (new) users' preferences as soon as possible by creating a short but informative questionnaire. In this section we explain the core of our contribution which is to use the prediction of available ratings to enhance active learning using decision trees.

Current decision trees techniques exploit only the available ratings in R in order to (1) find the discriminative items, (2) split the users' population, and (3) compute predictions over the candidate items. For instance, let us fix the decision trees analysis into the node t. These techniques iterate among items in I_t and analyze their impact in R_t. This impact is typically measured by associating an error when splitting users using their preferences to these candidate items. The goal is to find out, for each node of the tree, the best discriminative item that optimizes this error.

In addition, these techniques use a simple item prediction method based on the "item rating average" in order to evaluate a prediction accuracy and to compute prediction labels for candidate items. Note that this technique is adequate for large datasets, which allows a quick and acceptable generation of predictions from the available ratings in R_t. On the other hand, using more accurate prediction techniques is possible but (1) it is very expensive and time consuming to do it for every node of the tree, and (2) the predictions

needed in decision trees are item-oriented regardless of the user (the same item prediction value to any user) rather than user-item oriented (items' predictions depend on users' interests).

We propose to change this paradigm by using more accurate predictions over the available ratings R. The main idea is to introduce the prediction P as a new source of useful data. Hence, R and P are available from the root node of the tree. Then, when the node is split into child nodes, R_t is split into $R_{t-child}$. As long as we want to preserve that for every rating $r_{u,i} \in R_t$ there is an associated prediction $p_{u,i} \in P_t$, for every node t we split P_t into $P_{t-child}$ as well. This idea is illustrated in Figure 6.3. In addition, we propose to use the available ratings in R only to split the users population, and P to find out the best discriminative items to enhance the prediction label of candidate items.

This makes sense since finding discriminative items and label predictions are associated with computing an error. As long as P is built by using more accurate methods than the "item rating average", this error is minimized efficiently. We propose using efficient algorithms, such as matrix factorization [Zhou et al., 2008]. The main drawback of using matrix factorization is that it computes different item predictions for different users. The decision trees require a unique item prediction value to be applied to any user. In [Golbandi et al., 2011, Karimi et al., 2015b] the authors use the "item rating average" within R_t. We suggest using a similar method, with a major difference that is computing the "item prediction average", which is indeed the average of the predictions within P_t.

Collaborative filtering methods are very accurate for recommending items to users by replicating the users' rating behavior. As a consequence, they are good as well in guessing the average prediction of users, items, and in general the average rating value of the dataset. We illustrate this by using a real use-case. We perform the matrix factorization (MF) predictions over the Movielens 1M dataset in Table 6.3 to obtain the predicted values P for R. Table 6.2 summarizes statistics over R and P. One can see how close the statistic values such as the mean and quartiles between the available ratings and the predicted ratings are.

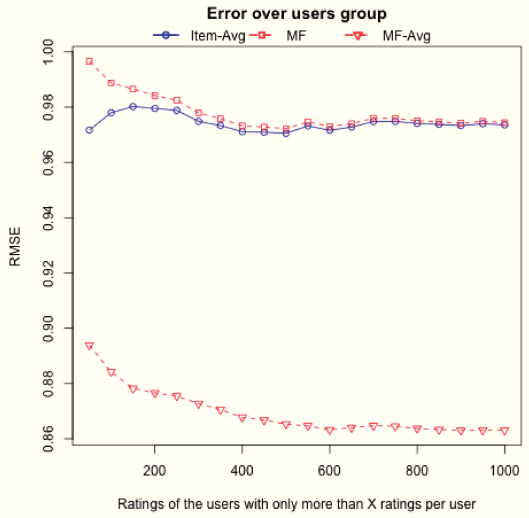
In addition, we want to go further by considering different users' populations. We compare the performance of three different predictors: (1) the item rating average (Item-Avg),

Table 6.2: Statistics for available ratings and matrix factorization predictions. MF1 and MF2 denote predictions over Movielens 1M and Movielens 10M, respectively.

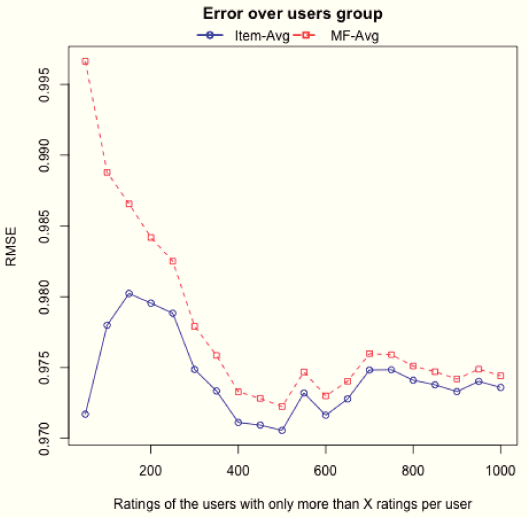
Statistic	MovieLens 1M	MF1	MovieLens 10M	MF2
1st Quartile	3.00	3.18	3.00	3.13
Median	4.00	3.66	4.00	3.58
Mean	3.58	3.58	3.51	3.51
3rd Quartile	4.00	4.05	4.00	3.96

(2) the matrix factorization (MF), and (3) the matrix factorization average (MF-Avg). The first sets the item's rating average as the item's prediction. The second decomposes a matrix R into two random matrices in such a way that the multiplication of both matrices gives approximately the original R [Zhou et al., 2008]. The third uses a matrix of predictions over available ratings given by the matrix factorization to set the item's prediction average as the item's prediction. Our goal is to show that (1) as demonstrated, the matrix factorization performs better in terms of accuracy than the item rating average method, and (2) the item prediction average predictions are close to the item rating average predictions.

This analysis is performed as follows. First, we obtain P by performing the matrix factorization over the available R. Second, we count the number of ratings per user and we divide the users into groups; users having less than 50 ratings, less than 100 ratings, etc. Highlight that these groups are incremental and thus one user may belong to more than one group. For each of these groups, the users' ratings and the corresponding users' predictions are taken from R and P into separated sets R' and P'. Third, we evaluate the performance of predictions in P' by computing the root mean square error (RMSE). Moreover, we perform the item rating average over R', and item prediction average (MF-Avg) over P', in order to observe their performance as well. The results are given in Figures 6.4(a) and 6.4(b). As expected, MF outperforms the item rating average method. In addition, we observed that MF-Avg attempts to imitate the behavior of the item rating average method. Hence, the item prediction average is also an acceptable predictor regarding the item rating average. In fact, one can observe that item rating average and item prediction average are closer while dealing with more users and ratings. This behavior is perfect in decision trees since the top of the tree is more populated by users and ratings. On they contrary, they



(a) Accuracy of predictors.



(b) Zoom over Figure 6.4(a). The average predictors.

Figure 6.4: Prediction techniques and average comparisons regarding the RMSE.

diverge while dealing with less users and ratings. This means the accuracy in larger decision trees will be decreased. As long as active learning aims to create short questionnaire this has not a great impact on our approach.

6.4.3 The decision trees algorithms

This sections presents the algorithms used to develop our approaches for decision trees. We can apply our contribution to non supervised and supervised decision trees techniques. Their main drawback is the heavy analysis needed to find out the best discriminative item. For instance, given the current node t, these techniques iterate over all candidate items $i \in I_t$ by analyzing users' ratings on i. The users populations are then grouped into users' who rated item i and users who did not. Typically, the latter is more populated due to the sparse nature of the available dataset. Furthermore, the users who rated item i can be grouped into further categorizations, e.g. users who liked/hated, or who rated '1, 2, 3, 4, 5'. Then, the population of users in these nodes, and their ratings, are used to evaluate the performance of choosing i as one discriminative item of R_t.

The difference between supervised and non-supervised approaches is that the former as-

sumes that warm-users can be considered as cold-user from which some ratings are known, and hence, the known users' ratings can be used to validate the technique. As a consequence, it is possible to compute an error based on the prediction accuracy, such as RMSE. On the contrary, since non supervised techniques do not have any validation, they compute a statistical error based on the available ratings in the node. Nevertheless, in both approaches a validation is not possible in the 'unknown' nodes, since by definition, there is no rating label for these users to this item. As a consequence, the statistical error is mandatory in this case. Our approach uses similar statistics as [Golbandi et al., 2011].

6.4.3.1 Non Supervised Decision Trees for Active Learning

In [Golbandi et al., 2011], the authors define a set of statistics and an internal error using these statistics to find out the best discriminative item. In this approach, the best item is the one which reduces this error. In addition, as long as the tree nodes contain many ratings, they use the item rating average method to compute item label predictions for items.

We suggest to use the predictions P over the available ratings in R in order to enhance this technique. Formally, given the node t, there are ratings $r_{u,i} \in R_t$ so that $u \in U_t$ and $i \in I_t$. In addition, for each rating $r_{u,i}$ there is an associated prediction $p_{u,i} \in P_t$. The goal is to find out, among all candidate items $j \in I_t$, the best discriminative item i^*. To do that, we compare the current state of the node t to the impact of picking j as discriminative item, as it is explained below.

On the one hand, the current status of the node j is given by all the items' ratings in P_t. The properties of this status are given by the statistics presented in 6.1, which consider separately all items in the node. This allows to glimpse a current state error as 6.2, which is indeed the error contribution of every item j in the node t: $e^2(t)_j = sum^2(t)_j - (sum(t)_j)^2/n(t)_j$.

$$\forall j \in I - i : \quad sum(t)_j = \sum_{u \in U_t \cap P(j)} p_{u,j} \quad (6.1)$$
$$sum^2(t)_j = \sum_{u \in U_t \cap P(j)} p_{u,j}^2$$
$$n(t)_j = |u \in U_t \cap P(j)|$$

$$e^2(t) = \sum_j sum^2(t)_j - (sum(t)_j)^2 / n(t)_j \quad (6.2)$$

On the other hand, we look for the best candidate item that minimizes this current error. Thus, we analyze the impact of every item j within this node t and the child nodes. We first suppose that j is a discriminative item and we split U_t into 3 childs: tL for users who liked j ($r_{u,j} \geq 4$), tD for users who did not like ($r_{u,j} \leq 4$)) and tU for users who do not know the item (absence of rating). Highlight that we use R_t to split the users' population. In addition, we delete the presence of discriminative item ratings in child nodes. As a consequence, one has 3 rating subsets R_{tL}, R_{tD} and R_{tU} in child nodes U_{tL}, U_{tD} and U_{tU}, respectively. The impact of picking j as candidate item is given by the error of these three nodes: $Err_t(j) = e^2(tL) + e^2(tD) + e^2(tU)$. This means that one evaluates the status of the given child nodes by using the Equation 6.2 and associated subsets P_{tL}, P_{tD} and P_{tU}. However, the number of users in the unknown node tU is normally much larger than in other groups, and hence, the computation is heavier in this node. To avoid this, it is possible to deduce statistics for tU from the node t and the other child nodes tL and tD, as follows:

$$sum(t_u)_j = sum(t)_j - sum(t_L)_j - sum(t_D)_j \quad (6.3)$$
$$sum^2(t_u)_j = sum^2(t)_j - sum^2(t_L)_j - sum^2(t_D)_j$$
$$n(t_u)_j = n(t)_j - n(t_L)_j - n(t_D)_j$$

By doing this analysis with every candidate item $j \in I_t$, one obtains an associated impact $Err_t(j)$. The discriminative item is the one which minimizes such error, $i^* =$

$argminErr_t(j)$, and then it is used as the real splitter to continue the construction of the tree in lower levels.

Finally, once the discriminative item is chosen, the prediction label associated to the tree is given by the item prediction average, i.e. average value of $P_t(i^*)$.

This approach is similar to [Golbandi et al., 2011], with two major differences. First, the available ratings are only used to split the population of users. As a consequence, the statistics and the items predictions are computed by using the proposed set of predictions P. Second, once a discriminative item is chosen in a parent node it does not pass to the child nodes. This is done for two reasons: (1) to avoid to choose the same item, and hence, to avoid to pose twice the same question to the same user, and (2) to delete the influence of the items' ratings in the child nodes. In fact, one can avoid choosing an item without deleting their ratings as done in [Golbandi et al., 2011]. Algorithm 5 shows this approach.

6.4.3.2 Supervised Decision Trees for Active Learning

In [Karimi et al., 2015b], the authors suggested using warm-users as cold-users from whom some interests are known. This assumption allows to create a supervised decision trees where some labels are known for validation purposes.

We again suggest to use the predictions P over the available ratings in R in order to enhance this technique. Given the current node t, there are warm users in U_t who have ratings in R_t and predictions in P_t. This technique considers warm users as cold-users from who some ratings are known. Thus, we split the users ratings into training ratings $R_{t-train}$ and validation ratings $R_{t-validation}$. The former represents the interests of the warm users. The latter are the items labels given when considered as cold-users. Note that we consider the perfect case where a user u and and item j and in $U_{t-train}$, $U_{t-validation}$ and $I_{t-train}$, $I_{t-validation}$ respectively. As long as for each rating $r_{u,i}$ there is an associated prediction $p_{u,i} \in P_t$, it is possible to obtain $P_{t-train}$ as well.

As a result, the current node t contains ratings in $R_{t-train}$, predictions in $P_{t-train}$, and ratings for validation purposes in $R_{t-validation}$. The goal is to find out, among all candidate

Algorithm 5 Non-supervised decision tree algorithm

```
1: function BuildDecisionTree(R_t, P_t, currentTreeLevel)
2:     for rating r_{u,i} in R_t do
3:         accumulate statistics for i in node t using p_{u,i}
4:     end for
5:     for candidate item j in I_t do
6:         for r_{u,j} in R_t(j) do
7:             obtain P_t(u)
8:             split U_t into 3 child nodes based on j
9:             find the child node where u has moved into
10:            for rating p_{u,i} in P_t(u) do
11:                accumulate statistics for i in node t − child using p_{u,i}
12:            end for
13:        end for
14:        derive statistics for j in node tU from the tL and tD statistics
15:        candidate error: e_t(j) = e_tL(j) + e_tD(j) + e_tU(j)
16:    end for
17:    discriminative item i* = argmin_i e_t(i)
18:    compute p_{i*} by using item prediction average
19:    if currentTreeLevel < maxTreeLevel then
20:        create 3 child nodes U_{t−child} based on i* ratings
21:        for child in child nodes do
22:            exclude i* from R_{t−child}
23:            BuildDecisionTree( R_{t−child}, P_{t−child}, currentTreeLevel +1 )
24:        end for
25:    end if
26:    return i*
27: end function
```

items $j \in I_t$, the best discriminative item i^*. To do that, we compare the current state of the node t to the impact of picking j as discriminative item, as it is explained below.

On the one hand, the state of the node t is given by the Root Mean Square Error (RMSE), which computes the squared difference between current predictions and real observed ratings. Predictions are computed by using the "item predicion average" method over $P_{t-train}$. Observed ratings are taken from $R_{t-validation}$. Therefore, each user $u \in U_t$ is associated to one error $RMSE1_u$. The current error in the node is the sum of all user's errors.

On the other hand, we look for the best candidate item that minimizes this current error. Thus, we analyze the impact of every item j within this node t and the child nodes.

We first suppose that j is a discriminative item and we split U_t into 3 childs: tL for users who liked j ($r_{u,j} \geq 4$), tD for users who did not like ($r_{u,j} \leq 4$)) and tU for users who do not know the item (absence of rating). Highlight that we use $R_{t-train}$ to split the users' population. In addition, we delete the presence of discriminative item ratings in child nodes. As a consequence, one has 3 rating subsets $R_{t-train-L}$, $R_{t-train-D}$ and $R_{t-train-U}$ in child nodes $U_{t-train-L}$, $U_{t-train-D}$ and $U_{t-train-U}$, respectively. Hence, it is possible to compute the state of the child nodes as it was performed before. This creates a second user error, $RMSE2_u$. The impact of picking j as candidate item for the user u is $\triangle_{u,i}= RMSE1_u - RMSE2_u$. Thus the impact of picking j as candidate item for the node t is given by the sum aggregation of all impacts of j.

The number of candidate items to analyze can be very large. We take into account only the 200 most popular items, as done in [Karimi et al., 2015b]. This yields in very acceptable accuracy and a great reduction in the time analysis.

Therefore, after the analysis of all users and candidate items, the item in $R_{t-train}$ associated to a higher overall impact is picked as discriminative item i^*, since the maximum value here means a higher error difference. Finally, the predicted label for this discriminative items is given by the item prediction average over current node P_t.

This approach is similar to [Karimi et al., 2015b], with two major differences: (1) P is used to validate the approach, and to obtain items label for the chosen discriminative items, and (2) we split the nodes into 3 child nodes ('like', 'dislike', 'unknown') rather than 6. This warrants a minimum number of users in child nodes to avoid fast pruning in small datasets. Algorithm 6 shows this approach.

6.4.4 Complexity of the algorithm and Time analysis

The complexity of our approaches for non-supervised decision trees and supervised decision trees is very similar to [Golbandi et al., 2011, Karimi et al., 2015b]. In fact, despite some differences in the way of computing the error that needs to be improved, these algorithms follow a similar procedure. The complexity of splitting the users in node t is $O(\sum_{u \in U_t} |R_t(u)|^2)$, and thus, for all the nodes in the same level we use $O(\sum_{u \in U} |R(u)|^2)$. As a consequence, the complexity to build a tree of N questions is $O(N \sum_{u \in U} |R(u)|^2)$.

Algorithm 6 Supervised decision tree algorithm

function BUILDDECISIONTREE(U_t, $R_{t-train}$, $R_{t-validation}$, P_t, *currentTreeLevel*)
  for user $u \in U_t$ **do**
    compute $RMSE_u^1$ on $R_{t-validation}(u)$ and $P_t(u)$
  end for
  for candidate item j from $R_{t-train}$ **do**
    split U_t into 3 child nodes based on j
    for user $u \in U_t$ **do**
      find the child node where u has moved into
      compute $RMSE_u^2$ on $R_{t-validation}(u)$ and $P_t(u)$
      $\triangle_{u,i} = RMSE_u^1 - RMSE_u^2$
    end for
  end for
  δ = aggregate all $\triangle_{u,i}$
  discriminative item $i^* = argmax_i \delta_i$
  compute p_{i^*} by using item prediction average
  if currentTreeLevel $<$ maxTreeLevel and $\triangle_{i^*} \geq 0$ **then**
    create 3 child nodes $U_{t-child}$ based on based on i^* ratings
    for *child* in child nodes **do**
      exclude i^* from $R_{t-child}$
      BuildDecisionTree($U_{t-child}$, $R_{t-child-train}$, $R_{t-child-validation}$, $P_{t-child}$, *currentTreeLevel +1*)
    end for
  end if
  return i^*
end function

In fact, adding the prediction set P does not affect the complexity of the algorithms, although, it does affect the memory footprint of the approaches. Considering that ratings R and predictions P sets are coded equally, our approach consumes double of the memory size to store the set P. In addition, extra runtime is required to split both R and P accordingly.

The time-consumption of the algorithms is similar as well. A little extra time is needed in our approaches in order to deal with the split of the prediction in P. In addition, in comparison to [Karimi et al., 2015b] our supervised approach is faster due to the reduced number of child nodes.

Table 6.3: Properties of different movie datasets.

Property	MovieLens1M	MovieLens10M	Netflix
Users	6040	71567	480000
Items	3900	10681	17000
Ratings	1 million	10 millions	100 millions
Sparsity	0,042%	1,308%	1,225%
Scale	Integer 1-5	1-5 by 0.5	Integer 1-5

6.5 Experimentation

The goal of our experimentation is two-fold (i) to present the behaviour of current techniques in smaller datasets and (ii) to show the performance of our presented approach. Recent techniques have presented their results using Netflix dataset. However, this dataset is no longer available for research. Hence, we use two versions of the Movielens dataset. Table 6.3 describes the properties of these datasets.

Since our approach considers external techniques prediction as a new source, in order to build our decision trees we use matrix factorization [Zhou et al., 2008] due to its accuracy. We compare our approach in non supervised decision trees, as in [Golbandi et al., 2011], and in supervised decision trees, as in [Karimi et al., 2015b]. The latter will not be largely explained due to a lack of space.

The technique in [Golbandi et al., 2011], denoted "Golbandi", uses two parameters: bias overfitting λ_1 and rating overfitting λ_2. We set empirically these parameters to $\lambda_1 = 7$ and $\lambda_2 = 200$. The technique in [Karimi et al., 2015b], denoted "Karimi", uses a rating overfitting parameter λ_2. Our approaches are identified as "Pozo Non Supervised" and "Pozo Supervised", and they do not contain any parameters, which represents an advantage in comparison to the state of the art.

As long as non supervised decision trees and supervised decision trees require different settings, we explain these experimentations separately. In order to compare the approaches we use the RMSE metric oriented to users, which measures the squared difference between the real ratings and the predicted ratings:

$$RMSE_u = \sqrt{\frac{1}{N}\sum(r_{u,i} - p_i)^2}$$

Where N is the number of ratings of the user u, p_i is the predicted label value of the candidate item in the question node and $r_{u,i}$ is the real rating of the user u for the item i. Hence, the evaluation of the error in one question is the average of the users error in this question number. As a consequence, for this metric the lower is the better.

Knowing that the experimentation may depend on the split of the dataset, we run it 50 times and then used the mean value of the RMSE. We use this process to evaluate the performance for the MovieLens 1M and MovieLens 10M.

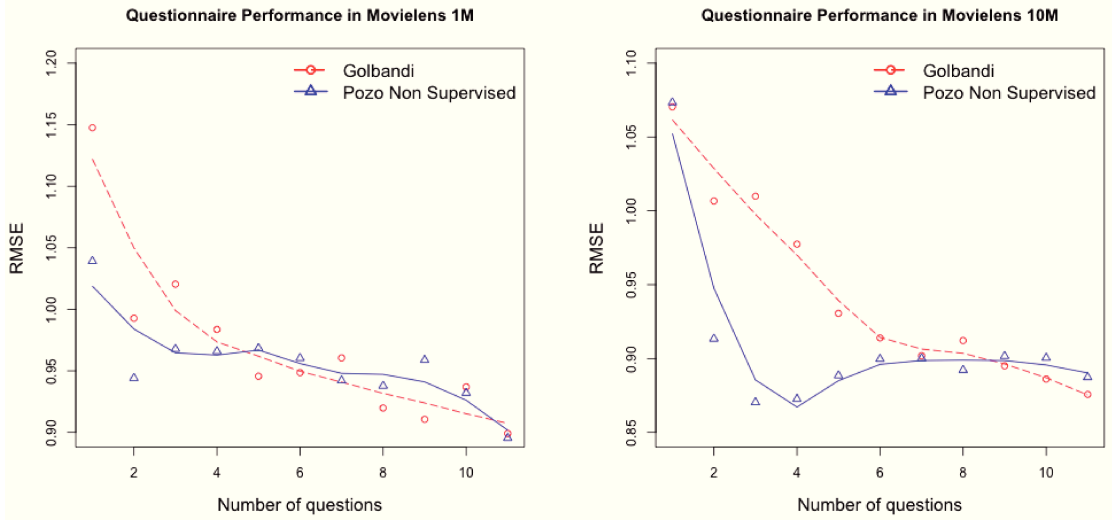
6.5.1 Non supervised decision trees

The experimentation carried out in [Golbandi et al., 2011] splits the datasets into 90% training set, D_{train} and 10% test set, D_{test}. However, this is not a real cold-start context since the same user may appear in both training and test set. We suggest a real cold-start situation. We split the set of users in the datasets into 90% training set, U_{train} and 10% test set, U_{test}. Hence, the users in the training set help to build the decision trees and the users in the test set are considered as new user to evaluate the performance of the approach.

The process we have followed to run this experimentation is as follows. First we split the dataset into U_{train} and U_{test}. Second, we compute the collaborative filtering algorithms over ratings R in U_{train} and we extract the associated predictions P. Third, we train the approach of "Golbandi" by using U_{train}. Our approach is trained by using both ratings in training set R and the prediction of the training set P. Finally, the performance of the decision trees is evaluated by using the test set U_{test}. The users in this set are used to answer the questions. If the item is known, we compute the RMSE associated to this answer and this question. Then, the user answers a new question. At the end, we compute the average of the accumulated nodes RMSE.

Figures 6.5(a) and 6.5(b) show the results (the mean point values and tendency curves) of this experimentation for MovieLens 1M dataset and MovieLens 10M dataset respectively.

On the one hand, our approach achieves a lower error in less number of questions. This



(a) MovieLens 1M dataset.

(b) MovieLens 10M dataset.

Figure 6.5: Questionnaire performance in RMSE.

matches with the needs of active learning; short but very informative questionnaires. This is possible due to the higher accuracy of the matrix factorization. In addition, notice that approaches tend to an asymptotic behavior in both Movielens datasets. This corresponds to a case in which users are very profiled, and further knowledge is complicate to extract.

On the other hand, it is interesting to comment the particularity in Figure 6.5(b). "Golbandi" slowly decreases the error and starts finding the asymptote behavior at question 11. Our approach gives very informative questions at the beginning, making the error to decrease very fast. Further than question 4, our approach proposes less informative questions and thus the error slightly increases between the questions 6 and 8. The algorithm is looking for new more informative questions than before, what makes the error decreases again from questions 9, and to slowly meet the asymptote.

6.5.2 Supervised decision trees

The experimentation carried out in [Karimi et al., 2015b] use a 4-fold set to evaluate their dataset. The Netflix dataset is already split into 90% training set, D_{train}, 10% test set, D_{test}. In order to evaluate their technique in cold-start situations, they merge both

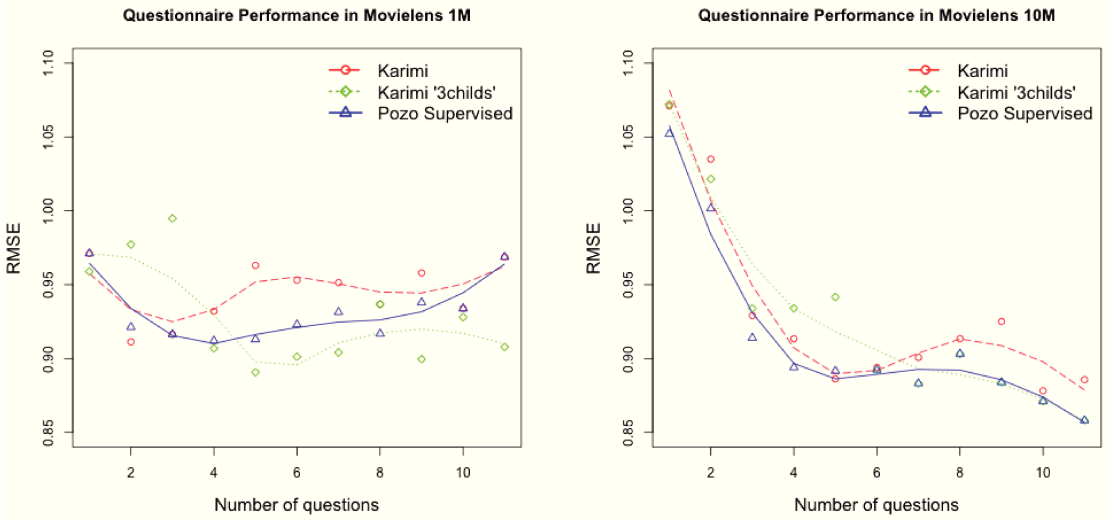
sets and split the result into 75% user training set, U_{train} and 25% user test set U_{test}, where users in one set are not present in the other. Then, they cross these 4 sets to create their own settings environment. To train the decision trees they intersect D_{train} and U_{train} to get a training set R_{train}. They use the intersection of D_{test} and U_{train} to obtain the validation set $R_{validation}$. To evaluate the technique, they use the intersection of D_{train} and U_{test} as the answer set R_{answer}, whereas the intersection of D_{test} and U_{test} is the performance set $R_{performance}$ used to evaluate the RMSE performance of the technique.

"Karimi" chose these experimentation settings in order to publish and compare results using the pre-defined D_{train} and D_{test} Netflix dataset. In addition, we consider that picking up 25% of users the for test set, U_{test} is concerned with the posteriori search of R_{answer} and $R_{performance}$. The more users in the test set the more probability to pick up ratings from D_{train} and D_{test} for R_{answer} and $R_{performance}$, respectively. Thus, more test points are possible possible to use.

We simplify the setting above. The process we have followed in the Movielens datasets is as follows. We split the dataset into 90% user training set, U_{train} and 10% user test set U_{test}, where users in one set are not present in the other. Then, the ratings in U_{train} are split into 90% rating training set, R_{train}, and 10% ratings validation set $R_{validation}$. The U_{test} is not split into "answer" and "performance" and it is completely used for probing the approach.

We train "Karimi" by using these settings. In addition, we train a modification of "Karimi" to take into account only 3 child nodes, denoted "Karimi 3 childs". Figures 6.6(a) and 6.6(b) show the results (the mean values and tendency curves) of this experimentation for both MovieLens datasets.

On the one hand, one may observe that our approach achieves a lower error (around 1%) in less number of questions; hence we better capture the preferences of new users. Therefore, our approach still matches with the needs of active learning: short but very informative questionnaires. This is possible due to the higher accuracy of the matrix factorization based predictions. In addition, it is expected to have an asymptotic behavior in large number of questions for both Movielens datasets.



(a) MovieLens 1M dataset.

(b) MovieLens 10M dataset.

Figure 6.6: Questionnaire performance in RMSE.

On the other hand, highlight the non-stability of "Karimi" in Figure 6.6(a). Karimi splits parent nodes into 6 child nodes per level; hence the Movielens 1M dataset has not enough users and ratings in deeper nodes of the tree, and thus it causes this dysfunction. The Karimi "3 childs" implementation enhances this issue by aggregating users into 3 child nodes ("like", "dislike" and "unknown"). However, our approach overcomes both approaches above. Highlight that the Karimi modification and our approach show close results. In fact, the difference is too small to be perceived in Figure 6.6(b). In addition, one may observe an increase of the error after 6 questions due to the smaller number of users and, again, the fast previously found informative questions.

6.5.3 Time analysis

We conclude the experimentations by comparing the time analysis within the different approaches. We use a virtual machine in the cloud with 6 cores Intel(R) Xeon(R) CPU E5-2650 at 2.00GHz and 14 Gb RAM. The techniques are implemented using a Java multi-threading approach, which distributes the analysis inside the tree nodes. Our non supervised approach increases the time consumption of Golbandi by around 1%, which might be considered as insignificant. On the contrary, the supervised approach reduces the

time consumtion of Karimi by around 50% due to the reduced number of child taken into consideration.

6.6 Discussion

Recommender Systems suffers from new user cold-start. This issue is more acute in collaborative filtering techniques since they only rely on users' ratings to generate recommendations. In order to learn the (new) users' preferences, passive recommender systems wait for sporadic ratings from users. On the contrary, active learning is proposed as a data acquisition method that gathers users' ratings by presenting a simple questionnaire to users. For instance, 'Do you like this movie?': 'Yes, I do like'; 'No, I do not like'; 'I have not seen it'. However, users are not willing to answer many questions or to rate many items. Therefore, the main goal of active learning is to create a short but efficient and informative questionnaire.

The state of the art has evolved from non personalized batch-oriented techniques (giving many questions at once and the same questions to any user) to personalized sequential-oriented questionnaires (giving questions one by one taking into account the past current users' answers). In our point of view, the personalization of the active learning technique is crucial to better learn the new users preferences and current methods based on decision trees are interesting techniques to model questionnaires. Indeed, the personalization of decision trees allows to predict with which items the new user has been in contact, and present them to the users.

However, active learning techniques based on decision trees have not been applied to small datasets. Less number of users and ratings affect the performance of these algorithms. In addition, we consider that recent approaches do not properly consider the prediction of candidate items. On the one hand, they use very simple prediction methods to make the decision trees tractable. On the other hand, they only exploit the users' ratings.

In this chapter we proposed using two sources in the decision trees active learning technique. The main idea is to train an accurate collaborative filtering techniques with a ratings dataset to generate a prediction dataset. Both ratings and predictions dataset

are used inside the decision trees. The former properly split the users' population while building the tree. The latter enhances the seek of the best discriminative items (questions) and better predict the associated labels. We have tested this approach in non supervised decision trees and supervised decision trees techniques.

The experimentation uses two publicly available datasets: Movielens 1 million ratings and Movielens 10 million ratings. We show that our approach enhances the state of the art in terms of RMSE and the related number of questions. In fact, we find better questions and better predictions that make it quicker reduce the error. This is especially useful in small dataset contexts, where the low number of users and ratings do not allow to create big decision trees.

Finally, the approach suggested in this chapter allows to alleviate the new user cold start, issue that, together with the scalability and large items/users representations, is a big challenge in the domain of recommender systems.

Chapter 7

Conclusions and Perspectives

7.1 Conclusions

Recommender systems help users to deal with information overload and in the decision making processes. They aim to reduce the huge amount of information that users of the Internet have to face by studying their interests and presenting first the information in which they may be more interested [Kantor et al., 2011]. This has raised the attention of the e-commerce and services providers in order to personalize the products, the services and the users' experience.

In Chapter 2 we have presented some techniques and approaches coming from the state of the art in academia and industry environments. The collaborative filtering methods are one of the most widely used approaches [Kantor et al., 2011]. Particularly, the matrix factorization techniques have been very well received in the recommendation field due to their great accuracy and scalability. However, these techniques do not exploit the heterogeneous data coming from items' descriptions or users' descriptions. In addition, the integration of this information into the matrix factorization process is complex and it affects to the domain dependency and scalability of the system. Another challenge of collaborative filtering techniques is the cold start [Su and Khoshgoftaar, 2009, Elahi et al., 2014]. This issue has been typically addressed in the state of the art by implementing hybrid recommender systems in order to alleviate the drawback of techniques with the advantages of the others.

This thesis has focused mainly on collaborative filtering approaches. We have chosen

matrix factorization as the baseline method. In the context of collaborative filtering we were interested in (1) the scalability of techniques and the distribution of them among multiple machines to alleviate the time analysis consumption, (2) discovering the interest of users in the attributes of items and integrate it in such a way that the domain dependency impact is reduced, (3) the representation of items and users through large dimension vectors, and (4) the new user cold start issue in active learning techniques based on collaborative filtering assumptions.

In Chapter 3 we have presented an analysis of the matrix factorization scalability and suggested one technique to distribute the recommendation process among multiple machines in a Hadoop/MapReduce cluster environment. Our matrix factorization technique uses stochastic gradient descent algebra to find users' interests and users' predictions. In order to parallelize the process, we decompose the matrix of ratings into blocks. Then, we define stratum as sets of blocks which are mutually independent (they do not share common users and items). Stratums can be computed parallely, where all blocks are analyzed by one different machine at every time. Our approach avoids the deletion of data compared to the state of the art [Makari et al., 2014], what slightly improves the recommendation accuracy.

The matrix factorization technique described in Chapter 3, a.k.a. Singular Value Decomposition for Recommender Systems [Funk, 2006], is used in this thesis as a baseline algorithm, for which we aim to perform more improvements. In Chapter 4 we proposed a new source of knowledge which is later exploited by the matrix factorization approach: the implicit interest of users in the attribute of items. In fact, the interest of users is often item-oriented [Oard et al., 1998], e.g. to rate a book, to watch a movie, to click on a video, etc. The users' interests in the items' attributes is difficult to capture due to the big quantity of possibilities. Our approach extracts this knowledge from past users' ratings and items' descriptions. The goal is to integrate it into matrix factorization techniques yet reducing the impact of domain dependency of the system. A few possibilities were discussed, and a layered architecture was chosen for our approach. We propose to transform existing ratings into "semantic ratings" that extends the sense of ratings to denote the interest of users in items and items' attributes. These new semantic ratings are then used in the matrix

factorization. The results of this proposition demonstrate a great improvement in rank-based recommendations, where it overcomes the precision, recall and f-measure metrics. The side effect of this approach is that top-K recommended items tend to be more similar, regarding an intra-similarity list metric.

In Chapter 5 we discussed that items and users can be described by very large amounts of data linked to items/users characteristics or other external resources such as social networks [Peis et al., 2008], [Kantor et al., 2011], [Dahimene et al., 2014]. Indeed, the number of features to describe items/users affects the correlation of data. Generally the more attributes are used the better the accuracy of the similarity is. Particularly similarity-based recommender systems (e.g. content-based and other hybrid approaches) represent items/users as a vector of features or keywords, and thus, vector similarity methods can be computed. Thus, these vectors can be very large, sparse and memory-consuming. To avoid this issue, the state of the art uses feature selection techniques. However, reducing the number of features may yield to a loss of quality in the item representation, and consequently, in items similarity. We proposed to use a compressed yet high quality data representation for items/users based on Bloom Filters [Broder and Mitzenmacher, 2004]. We exploit the bloom filters' properties in order to create two fast an accurate similarity measures: AND and XNOR. The former takes into account the common descriptions of items/users, which are inserted in their own filters. The latter takes into account both common insertions and common missing insertions. The most interesting of this approach is that filters consumes almost the 96% of the size of standard vector representations, and they achieve the 98% fidelity while computing similarities.

Our fourth challenge was to cope with cold-start issue. The new items cold start and the new users cold start are both present in recommender systems, although they are more acute in collaborative filtering techniques [Rubens et al., 2011], [Elahi et al., 2014]. Besides, the new items cold start can be more easily alleviated by taking into account the description of items, and thus using hybrid approaches. However the new user cold-start in collaborative filtering methods need to incrementally, adaptively and quickly find out the new users' preferences. In Chapter 6 we dealt with this issue by proposing an active learning technique based on collaborative filtering decision trees. In this method the

(new) users interact with the system that presents a personalized questionnaire, i.e. which changes depending on the users' answers. The goal is to retrieve informative answers which represent the users interest in order to adapt next questions and increase the knowledge about the users' preferences. Compared to other approaches, we suggest to exploit two sources of data: the warm users' ratings and the warm users' predictions over these ratings. Our approach is faster in understanding the users' preferences what allows to reduce the number of questions to pose.

Finally, the four aforementioned challenges and approaches are possible to be merged. One can easily imagine an active learning technique that captures the first users' preferences in order to slowly feed a matrix factorization technique which generate useful recommendations. In addition, the implicit interest of users in items' attributes can be taken into account in both active learning and matrix factorization. Furthermore, these interests are large descriptions of users through the items' descriptions, i.e. what a user like from items. This can be exploited by recommender systems in order to give explanatory recommendations (e.g. you may like this movie because it is a comedy and you liked the actors who play in it); and hence to increase the acceptance of recommendations. We are aware of the scalability of the system as well, and thus recommendation processes and memory-consumptions may be taken into account in the whole deployment of the recommender system.

The next section presents the perspectives and the on-going work of our research.

7.2 Perspectives

7.2.1 FIORA Project

In Chapter 1 we noted that this thesis is part of a collaborative project called FIORA[1], which aims to build a generic, robust and scalable recommender system. The domains in which this project will be applied are e-nutrition and e-tourism. As long as data from these domains were not available, we have experimented and published our contributions by using public datasets such as Movielens[2].

[1] www.fiora.pro
[2] http://grouplens.org/datasets/movielens/

Our first perspective focuses on testing our approaches under the context of FIORA, in particular the domain of e-nutrition. Indeed, one of the partners has recently launched a website called "My coach nutrition"[3], which aims to help users in the nutritional decisions in the every-day-life. It takes into account the users' profile and preferences, such as tastes and allergies, in order to suggest healthy and balanced diets and to teach the users to cook recipes and to correctly treat ingredients. Today, this site is open and data is being collected.

7.2.2 Positive and Negative implicit users' interests

Chapter 4 discussed the concept of the implicit users' interests in the items' attributes. We have seen that this interest comes from the number of occurrences of attributes (i.e. attributes' values, such as a particular movie genre or actor) in the past rated items. In addition, in order to be processed by collaborative filtering approaches, we transform ratings into "semantic ratings" by using a "semantic equation". However, this equation only provokes an addition in the original ratings.

The state of the art presents a special way of assuming whenever users like or not (binary assumption) an item depending on the rating the user gave to the item [Golbandi et al., 2011]. Thus, there is a more-less established threshold: ratings' values greater or equal to 4 (in a scale of 5) are considered to be liked and not liked in other case (assuming that 0 is the absence of rating and thus the absence of preference). It could be possible to exploit this assumption in order to find out positive and negative implicit users' interest in the items' attributes:

- Positive. The implicit positive interest of users' in the items' attributes is given by the number of occurrences of attributes over the past rated and liked items (rating greater or equal to 4 out of 5).
- Negative. The implicit negative interest of users' in the items' attributes is given by the number of occurrences of attributes over the past rated and disliked items (rating lower to 4 out of 5).

[3] www.mycoachnutrition.com/

As a consequence, it is possible to separately store and exploit this information. We point out that some attributes' values may appear in both positive and negative sets, as long as, for instance one actor can play on one movie liked by the user and one other movie not liked by the same user. However, since this information is implicit, the number of occurrences in these sets will be different. The new goal would be to find a new "semantic equation" to take both positive and negative interests into account. Hence, this equation would promote items containing positive implicit users' interest and would refrain the recommendation of negative implicit users' interest.

7.2.3 Large description of users through the items' descriptions

Some research works have proposed the "collaboration via content" [Pazzani, 1999] in which, users are correlated regarding the description of the items they rate, in addition to the typical rating-oriented correlations.

Chapter 4 suggests to extract the implicit interest of users in the items' attributes. This information is then inserted into bloom filters to be quickly available. However, in this chapter we have exploited bloom filters by simply querying it. Specially, we do not perform users' correlations or users' similarities regarding a possible "collaboration via content". On the other hand, in Chapter 5 the items' and users' bloom filter representations are useful to alleviate memory-consumption in large users' and items' descriptions. In addition, we proposed the AND and XNOR similarities among items/users. As a consequence, we consider that it can be interesting to merge both ideas in order to perform "collaboration via content" users' similarities by using the bloom filter representations and bloom filter similarity measures.

Furthermore, together with the perspective mentioned above, this would explain not only the positive and negative implicit interest of users (e.g. whether a user has special affinity to one actor), but also the positive and negative lack of interest (e.g. if a user has never been in contact with one actor he may like, or if a user has never been in contact with sportive content simply because he is not interested in it and never interacted with it).

7.2.4 Exploration, exploitation and explanation in recommender systems

The exploration paradigm consists in letting the user freely navigate among the items to discover and rate new unexpected items [Kantor et al., 2011]. On the contrary, the exploitation paradigm refers to the usage of the users' interest (explicit or implicit) and the persistence of recommendations, e.g. to continuously recommend the same items to users when these suggestions have not been taken in several occasions.

We consider that recommender systems would perform better recommendations by using very detailed descriptions of users and items [Mikeli et al., 2013b]. Thus, the exploration and exploitation trade-off will become more important, e.g. to let the users discover new actors and avoid to recommend movies where similar actors from movies he did not like also play. However, in our opinion, there is a third player in this trade-off: the explanation. The explanation paradigm allows the user to understand and accept the recommendations. In Chapter 4 we talked about the implicit interest in very detailed items' attributes, which are very difficult to capture in current systems. For instance, the interest of the user in the photography department of a movie. These details could become very interesting in the explanation of recommendations and would open other paths to understand the exploration and exploitation under very users' and items' detailed descriptions. The user could explore new items under a new regard (e.g. to watch a movie and evaluate the photography department). The system can exploit these new interactions accordingly to learn more about the users interests.

As a consequence, the assumptions in Chapters 4 and 5 can be useful to explain the recommendations: on the one hand, the preference of users in the attributes of items are known ("collaboration via content"); on the other hand, the bloom filter representation of user/items may allow to perform correlations exploiting the big information inserted in the bloom filters. The challenge is to develop a system which can exploit this information and compute detailed explanations about why users should accept recommended items based on the attributes that the users liked in the past.

7.2.5 Slope-One questionnaires for active learning

In Chapter 6 we have presented active learning techniques to tackle the new user cold start issue. These methods suggest to present a questionnaire to new users. The user is asked to rate some items and the system learns these first users' preferences [Elahi et al., 2014]. The goal is to find the best questions to pose in order to maximize the information gain of the system, i.e. quickly learn the users' preferences. Thus, these techniques look for the items which are informative and recognized by the user in order to retrieve as much information as possible.

Questionnaires should be efficient, short (low number of questions) and quickly react to users answers [Rubens et al., 2011, Elahi et al., 2014]. Due to these constraints, the current state of the art proposes to create the questionnaires off-line by using decision trees techniques [Golbandi et al., 2011]. As a result, the new user can answer first questions and new questions are presented immediately after. This methodology has demonstrated good results, however the current techniques use very simple prediction methods, such as the "item's rating average prediction", in order to build questionnaires in a reasonable time.

Better prediction for active learning methods can be used, such as the Slope-One methods that have demonstrated a better performance in cold-start situations and good prediction accuracy [Lemire and Maclachlan, 2005]. These methods are incremental, what allows very fast adaptation of the current models to the answers of users; they are scalable, what allows to parallelize the process among multiple machines; and they tend to recommend popular items first, what is very interesting in active learning contexts since they seek for recognizable items by users.

It is possible to create a Slope-One model based on the current warm users' ratings. This model would quickly vary together with the construction of the questionnaire creating an off-line questionnaire. In addition, it can incorporate new users' answers to the model making new questions more aware of the new users' preference.

Another perspective idea is to detect the new users preferences directly on the fly. We propose Slope-One methods (which are incremental and scalable) to adapt off-line decision tree questionnaire to on-line questionnaires.

7.2.6 Time-aware active learning techniques

Active learning techniques for collaborative filtering based on decision trees only take into account the past users' ratings to create questionnaires [Golbandi et al., 2011]. Then, the new users navigate through the questionnaires by answering (i.e. rating) questions (i.e. items).

We consider that there are other resources that appear in questionnaires and may be very useful and informative in collaborative filtering active learning techniques. We especially believe that the time (new) users spent to answer a question is very significant for extracting the users' preferences. Thus, we focus on "time-aware" recommendation techniques and decision trees to retrieve and exploit not only the users' answers but also the users' behavior. On the one hand, we study new Slope-One techniques to create time-aware off-line questionnaires due to their scalability and reactivity. On the other hand, other collaborative filtering techniques have already demonstrated their compatibility with timestamped preferences [Koren and Bell, 2011]. However, time-aware techniques tend to be timely expensive whereas active learning techniques require of fast adaptability to the current users' answers. As a result, the challenge is to create accurate, scalable recommendation models that evolves in real-time to cope with cold-start issues.

Bibliography

[Adomavicius and Kwon, 2007] Adomavicius, G. and Kwon, Y. (2007). New recommendation techniques for multicriteria rating systems. *Intelligent Systems, IEEE*, 22(3):48–55.

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

[Almeida et al., 2007] Almeida, P. S., Baquero, C., Preguiça, N., and Hutchison, D. (2007). Scalable bloom filters. *Information Processing Letters*, 101(6):255–261.

[Amatriain et al., 2009] Amatriain, X., Pujol, J. M., Tintarev, N., and Oliver, N. (2009). Rate it again: increasing recommendation accuracy by user re-rating. In *Proceedings of the third ACM conference on Recommender systems*, pages 173–180. ACM.

[Aranda et al., 2007] Aranda, J., Givoni, I., Handcock, J., and Tarlow, D. (2007). An online social network-based recommendation system. *Toronto, Ontario, Canada*.

[Arias et al., 2012] Arias, J. J. P., Vilas, A. F., and Redondo, R. P. D. (2012). Recommender systems for the social web.

[Baeza-Yates and Ribeiro-Neto, 1999] Baeza-Yates, R. A. and Ribeiro-Neto, B. (1999). Modern information retrieval. In *Modern Information Retrieval*, page 513, Boston, MA, USA. ACM, Addison-Wesley Longman Publishing Co., Inc.

[Balabanović, 1998] Balabanović, M. (1998). Exploring versus exploiting when learning user models for text recommendation. *User Modeling and User-Adapted Interaction*, 8(1-2):71–102.

[Balabanović and Shoham, 1997] Balabanović, M. and Shoham, Y. (1997). Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72.

[Barjasteh et al., 2015] Barjasteh, I., Forsati, R., Masrour, F., Esfahanian, A.-H., and Radha, H. (2015). Cold-start item and user recommendation with decoupled completion and transduction. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 91–98. ACM.

[Bellogín et al., 2010] Bellogín, A., Cantador, I., and Castells, P. (2010). A study of heterogeneity in recommendations for a social music service. In *Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, pages 1–8. ACM.

[Ben-Shimon, 2013] Ben-Shimon, D. (2013). Anytime algorithms for top-n recommenders. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 463–466. ACM.

[Berkovsky et al., 2007] Berkovsky, S., Kuflik, T., and Ricci, F. (2007). Distributed collaborative filtering with domain specialization. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 33–40. ACM.

[Bernardes et al., 2015] Bernardes, D., Diaby, M., Fournier, R., FogelmanSoulié, F., and Viennet, E. (2015). A social formalism and survey for recommender systems. *ACM SIGKDD Explorations Newsletter*, 16(2):20–37.

[Blei et al., 2003] Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022.

[Bloom, 1970] Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426.

[Boim et al., 2011a] Boim, R., Milo, T., and Novgorodov, S. (2011a). Direc: Diversified recommendations for semantic-less collaborative filtering. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1312–1315. IEEE.

[Boim et al., 2011b] Boim, R., Milo, T., and Novgorodov, S. (2011b). Diversification and refinement in collaborative filtering recommender. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 739–744. ACM.

[Bonhard and Sasse, 2006] Bonhard, P. and Sasse, M. A. (2006). 'knowing me, knowing you' - using profiles and social networking to improve recommender systems. *BT Technology Journal*, 24(3):84–98.

[Boutilier et al., 2002] Boutilier, C., Zemel, R. S., and Marlin, B. (2002). Active collaborative filtering. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pages 98–106. Morgan Kaufmann Publishers Inc.

[Braunhofer et al., 2015] Braunhofer, M., Fernández-Tobìas, I., and Ricci, F. (2015). Parsimonious and adaptive contextual information acquisition in recommender systems. *Proceedings of IntRS'15*.

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

[Bridge et al., 2005] Bridge, D., Göker, M. H., McGinty, L., and Smyth, B. (2005). Case-based recommender systems. *The Knowledge Engineering Review*, 20(03):315–320.

[Broder and Mitzenmacher, 2004] Broder, A. and Mitzenmacher, M. (2004). Network applications of bloom filters: A survey. *Internet mathematics*, 1(4):485–509.

[Brozovsky and Petricek, 2007] Brozovsky, L. and Petricek, V. (2007). Recommender system for online dating service. *arXiv preprint cs/0703042*.

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

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

[Cantador et al., 2011] Cantador, I., Brusilovsky, P., and Kuflik, T. (2011). 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011). In *Proceedings of the 5th ACM conference on Recommender systems*, RecSys 2011, New York, NY, USA. ACM.

[Caraciolo, 2012] Caraciolo, M. (2012). Crab.

[Carenini et al., 2003] Carenini, G., Smith, J., and Poole, D. (2003). Towards more conversational and collaborative recommender systems. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 12–18. ACM.

[Castells et al., 2011] Castells, P., Vargas, S., and Wang, J. (2011). Novelty and diversity metrics for recommender systems: choice, discovery and relevance. In *International Workshop on Diversity in Document Retrieval (DDR 2011) at the 33rd European Conference on Information Retrieval (ECIR 2011)*.

[Chatzicharalampous et al., 2015] Chatzicharalampous, E., Christos, Z., and Vakali, A. (2015). Exploriometer: Leveraging personality traits for coverage and diversity aware recommendations. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1463–1468. ACM.

[Chee et al., 2001] Chee, S. H. S., Han, J., and Wang, K. (2001). Rectree: An efficient collaborative filtering method. In *Data Warehousing and Knowledge Discovery*, pages 141–151. Springer.

[Chen et al., 2012] Chen, T., Zhang, W., Lu, Q., Chen, K., Zheng, Z., and Yu, Y. (2012). Svdfeature: A toolkit for feature-based collaborative filtering. *Journal of Machine Learning Research*, 13:3585–3588.

[Cherfi et al., 2016] Cherfi, H., Atemezing, G., Amardeilh, F., and Rouzé, F. (2016). Enrichissement d'une base documentaire pour un système de recommandation dans le tourisme. *I2D–Information, données and documents*, 53(2):60–62.

[Dahimene et al., 2014] Dahimene, R., Constantin, C., and du Mouza, C. (2014). Recland: A recommender system for social networks. In *Proceedings of the 23rd ACM International*

Conference on Conference on Information and Knowledge Management, pages 2063–2065. ACM.

[Davidson et al., 2010] Davidson, J., Liebald, B., Liu, J., Nandy, P., Van Vleet, T., Gargi, U., Gupta, S., He, Y., Lambert, M., Livingston, B., and Sampath, D. (2010). The youtube video recommendation system. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 293–296, New York, NY, USA. ACM.

[Dean and Ghemawat, 2008] Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

[EasyRec, 2013] EasyRec (2013). Easyrec.

[Ekstrand et al., 2011] Ekstrand, M. D., Ludwig, M., Kolb, J., and Riedl, J. T. (2011). Lenskit: a modular recommender framework. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 349–350. ACM.

[Elahi et al., 2014] Elahi, M., Ricci, F., and Rubens, N. (2014). Active learning in collaborative filtering recommender systems. In *E-Commerce and Web Technologies*, pages 113–124. Springer.

[Felfernig and Burke, 2008] Felfernig, A. and Burke, R. (2008). Constraint-based recommender systems: technologies and research issues. In *Proceedings of the 10th international conference on Electronic commerce*, page 3. ACM.

[Fernández et al., 2006] Fernández, Y. B., Arias, J. J. P., Nores, M. L., Solla, A. G., and Cabrer, M. R. (2006). Avatar: An improved solution for personalized tv based on semantic inference. *Consumer Electronics, IEEE Transactions on*, 52(1):223–231.

[Funk, 2006] Funk, S. (2006). Netflix update: Try this at home (3rd place).

[Gantner et al., 2011] Gantner, Z., Rendle, S., Freudenthaler, C., and Schmidt-Thieme, L. (2011). Mymedialite: A free recommender system library. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 305–308, New York, NY, USA. ACM.

[Gashler, 2011] Gashler, M. S. (2011). Waffles: A machine learning toolkit. *Journal of Machine Learning Research*, MLOSS 12:2383–2387. http://waffles.sourceforge.net.

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

[Geravand and Ahmadi, 2014] Geravand, S. and Ahmadi, M. (2014). An efficient and scalable plagiarism checking system using bloom filters. *Computers & Electrical Engineering*, 40(6):1789 – 1800.

[Golbandi et al., 2010] Golbandi, N., Koren, Y., and Lempel, R. (2010). On bootstrapping recommender systems. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1805–1808. ACM.

[Golbandi et al., 2011] Golbandi, N., Koren, Y., and Lempel, R. (2011). Adaptive bootstrapping of recommender systems using decision trees. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 595–604. ACM.

[Goldberg et al., 1992] Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70.

[Gonzalez et al., 2012] Gonzalez, J. E., Low, Y., Gu, H., Bickson, D., and Guestrin, C. (2012). Powergraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)*, Hollywood.

[Gopalani and Arora, 2015] Gopalani, S. and Arora, R. (2015). Comparing apache spark and map reduce with performance analysis using k-means. *International Journal of Computer Applications*, 113(1).

[Guo et al., 2010] Guo, D., Wu, J., Chen, H., Yuan, Y., and Luo, X. (2010). The dynamic bloom filters. *Knowledge and Data Engineering, IEEE Transactions on*, 22(1):120–133.

[Hahsler, 2011] Hahsler, M. (2011). recommenderlab: A framework for developing and testing recommendation algorithms. *Nov*.

[Hannon et al., 2010] Hannon, J., Bennett, M., and Smyth, B. (2010). Recommending twitter users to follow using content and collaborative filtering approaches. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 199–206. ACM.

[Harpale and Yang, 2008] Harpale, A. S. and Yang, Y. (2008). Personalized active learning for collaborative filtering. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 91–98. ACM.

[Herlocker et al., 2000] Herlocker, J. L., Konstan, J. A., and Riedl, J. (2000). Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 241–250. ACM.

[Hernando et al., 2013] Hernando, A., Bobadilla, J., Ortega, F., and GutiéRrez, A. (2013). Trees for explaining recommendations made through collaborative filtering. *Information Sciences*, 239:1–17.

[Hofmann, 2003] Hofmann, T. (2003). Collaborative filtering via gaussian probabilistic latent semantic analysis. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 259–266. ACM.

[Hu et al., 2008] Hu, Y., Koren, Y., and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. IEEE.

[Instituut/Novay, 2009] Instituut/Novay, T. (2009). Duine.

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

[Jain et al., 2005] Jain, N., Dahlin, M., and Tewari, R. (2005). Using bloom filters to refine web search results. In *WebDB*, pages 25–30.

[Jain et al., 2013] Jain, P., Netrapalli, P., and Sanghavi, S. (2013). Low-rank matrix completion using alternating minimization. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, pages 665–674. ACM.

[Järvelin and Kekäläinen, 2002] Järvelin, K. and Kekäläinen, J. (2002). Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446.

[Jin and Si, 2004] Jin, R. and Si, L. (2004). A bayesian approach toward active learning for collaborative filtering. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 278–285. AUAI Press.

[Kampffmeyer, 2015] Kampffmeyer, M. C. (2015). Parallelization of the alternating-least-squares algorithm with weighted regularization for efficient gpu execution in recommender systems.

[Kantor et al., 2011] Kantor, P. B., Ricci, F., Rokach, L., and Shapira, B. (2011). Recommender systems handbook. In *Recommender systems handbook*, page 848. Springer, Springer US.

[Karim, 2014] Karim, J. (2014). Hybrid system for personalized recommendations. In *Research Challenges in Information Science (RCIS), 2014 IEEE Eighth International Conference on*, pages 1–6. IEEE.

[Karimi, 2014] Karimi, R. (2014). Active learning for recommender systems. *KI-Künstliche Intelligenz*, 28(4):329–332.

[Karimi et al., 2011a] Karimi, R., Freudenthaler, C., Nanopoulos, A., and Schmidt-Thieme, L. (2011a). Active learning for aspect model in recommender systems. In *Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on*, pages 162–167. IEEE.

[Karimi et al., 2011b] Karimi, R., Freudenthaler, C., Nanopoulos, A., and Schmidt-Thieme, L. (2011b). Non-myopic active learning for recommender systems based on matrix factorization. In *Information Reuse and Integration (IRI), 2011 IEEE International Conference on*, pages 299–303. IEEE.

[Karimi et al., 2011c] Karimi, R., Freudenthaler, C., Nanopoulos, A., and Schmidt-Thieme, L. (2011c). Towards optimal active learning for matrix factorization in recommender systems. In *Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference on*, pages 1069–1076. IEEE.

[Karimi et al., 2012] Karimi, R., Freudenthaler, C., Nanopoulos, A., and Schmidt-Thieme, L. (2012). Exploiting the characteristics of matrix factorization for active learning in recommender systems. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 317–320. ACM.

[Karimi et al., 2015a] Karimi, R., Freudenthaler, C., Nanopoulos, A., and Schmidt-Thieme, L. (2015a). Comparing prediction models for active learning in recommender systems. In *Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB. October 2015*. http://ceur-ws.org.

[Karimi et al., 2015b] Karimi, R., Nanopoulos, A., and Schmidt-Thieme, L. (2015b). A supervised active learning framework for recommender systems based on decision trees. *User Modeling and User-Adapted Interaction*, 25(1):39–64.

[Karimi et al., 2014] Karimi, R., Wistuba, M., Nanopoulos, A., and Schmidt-Thieme, L. (2014). Factorized decision trees for active learning in recommender systems. In *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on*, pages 404–411. IEEE.

[Katz et al., 2011] Katz, G., Ofek, N., Shapira, B., Rokach, L., and Shani, G. (2011). Using wikipedia to boost collaborative filtering techniques. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 285–288. ACM.

[Kautz et al., 1997] Kautz, H., Selman, B., and Shah, M. (1997). Referral web: combining social networks and collaborative filtering. *Communications of the ACM*, 40(3):63–65.

[Kim and Li, 2004] Kim, B. M. and Li, Q. (2004). Probabilistic model estimation for collaborative filtering based on items attributes. In *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 185–191. IEEE Computer Society.

[Kim et al., 2004] Kim, M. W., Kim, E. J., and Ryu, J. W. (2004). A collaborative recommendation based on neural networks. In *International Conference on Database Systems for Advanced Applications*, pages 425–430. Springer.

[Knijnenburg et al., 2012] Knijnenburg, B. P., Willemsen, M. C., Gantner, Z., Soncu, H., and Newell, C. (2012). Explaining the user experience of recommender systems. *User Modeling and User-Adapted Interaction*, 22(4-5):441–504.

[Kohrs and Merialdo, 2001] Kohrs, A. and Merialdo, B. (2001). Improving collaborative filtering for new users by smart object selection. In *Proceedings of International Conference on Media Features (ICMF)*.

[Konstan and Riedl, 2012] Konstan, J. A. and Riedl, J. (2012). Recommender systems: from algorithms to user experience. *User Modeling and User-Adapted Interaction*, 22(1-2):101–123.

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

[Koren, 2009] Koren, Y. (2009). The bellkor solution to the netflix grand prize. *Netflix prize documentation*.

[Koren, 2010] Koren, Y. (2010). Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(1):1.

[Koren and Bell, 2011] Koren, Y. and Bell, R. (2011). Advances in collaborative filtering. In *Recommender Systems Handbook*, pages 145–186. Springer.

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

[Lakiotaki et al., 2008] Lakiotaki, K., Tsafarakis, S., and Matsatsinis, N. (2008). Uta-rec: a recommender system based on multiple criteria analysis. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 219–226. ACM.

[Lam and Riedl, 2004] Lam, S. K. and Riedl, J. (2004). Shilling recommender systems for fun and profit. In *Proceedings of the 13th international conference on World Wide Web*, pages 393–402. ACM.

[Lee et al., 2012] Lee, J., Sun, M., and Lebanon, G. (2012). A Comparative Study of Collaborative Filtering Algorithms. *ArXiv e-prints*.

[Lemire, 2003] Lemire, D. (2003). Cofi.

[Lemire and Maclachlan, 2005] Lemire, D. and Maclachlan, A. (2005). Slope one predictors for online rating-based collaborative filtering. In *SDM*, volume 5, pages 1–5. SIAM.

[Li et al., 2008] Li, Q., Wang, C., and Geng, G. (2008). Improving personalized services in mobile commerce by a novel multicriteria rating approach. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 1235–1236, New York, NY, USA. ACM.

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

[Liu, 2015] Liu, L. (2015). *Performance comparison by running benchmarks on Hadoop, Spark, and HAMR*. PhD thesis, University of Delaware.

[Liu et al., 2012] Liu, Q., Chen, E., Xiong, H., Ding, C. H., and Chen, J. (2012). Enhancing collaborative filtering by user interest expansion via personalized ranking. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 42(1):218–233.

[Low et al., 2010] Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J. M. (2010). Graphlab: A new framework for parallel machine learning. *arXiv preprint arXiv:1006.4990*.

[Ma et al., 2011] Ma, H., Zhou, D., Liu, C., Lyu, M. R., and King, I. (2011). Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 287–296. ACM.

[Mabroukeh and Ezeife, 2011] Mabroukeh, N. R. and Ezeife, C. I. (2011). Ontology-based web recommendation from tags. In *Data Engineering Workshops (ICDEW), 2011 IEEE 27th International Conference on*, pages 206–211. IEEE.

[Makari et al., 2014] Makari, F., Teflioudi, C., Gemulla, R., Haas, P., and Sismanis, Y. (2014). Shared-memory and shared-nothing stochastic gradient descent algorithms for matrix completion. *Knowledge and Information Systems*, pages 1–31.

[Manouselis and Costopoulou, 2007] Manouselis, N. and Costopoulou, C. (2007). Analysis and classification of multi-criteria recommender systems. *World Wide Web*, 10(4):415–441.

[Manouselis et al., 2013] Manouselis, N., Drachsler, H., Verbert, K., and Duval, E. (2013). Recommender systems for learning.

[Massa and Avesani, 2004] Massa, P. and Avesani, P. (2004). Trust-aware collaborative filtering for recommender systems. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 492–508. Springer.

[Masthoff, 2011] Masthoff, J. (2011). Group recommender systems: Combining individual models. In *Recommender systems handbook*, pages 677–702. Springer.

[McPherson et al., 2001] McPherson, M., Smith-Lovin, L., and Cook, J. M. (2001). Birds of a feather: Homophily in social networks. *Annual review of sociology*, pages 415–444.

[Mello et al., 2010] Mello, C. E., Aufaure, M.-A., and Zimbrao, G. (2010). Active learning driven by rating impact analysis. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 341–344. ACM.

[Menezes et al., 2013] Menezes, D., Lacerda, A., Silva, L., Veloso, A., and Ziviani, N. (2013). Weighted slope one predictors revisited. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 967–972. ACM.

[Meyer, 2012] Meyer, F. (2012). Recommender systems in industrial contexts. *arXiv preprint arXiv:1203.4487*.

[Mi and Xu, 2011] Mi, Z. and Xu, C. (2011). A recommendation algorithm combining clustering method and slope one scheme. In *Bio-Inspired Computing and Applications*, pages 160–167. Springer.

[Middleton et al., 2004] Middleton, S. E., Shadbolt, N. R., and De Roure, D. C. (2004). Ontological user profiling in recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):54–88.

[Mikeli et al., 2013a] Mikeli, A., Apostolou, D., and Despotis, D. (2013a). A multi-criteria recommendation method for interval scaled ratings. In *Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2013 IEEE/WIC/ACM International Joint Conferences on*, volume 3, pages 9–12.

[Mikeli et al., 2013b] Mikeli, A., Sotiros, D., Apostolou, D., and Despotis, D. (2013b). A multi-criteria recommender system incorporating intensity of preferences. In *Information, Intelligence, Systems and Applications (IISA), 2013 Fourth International Conference on*, pages 1–6.

[Mobasher et al., 2004] Mobasher, B., Jin, X., and Zhou, Y. (2004). Semantically enhanced collaborative filtering on the web. In *Web Mining: From Web to Semantic Web*, pages 57–76. Springer.

[Narayanan and Shmatikov, 2008] Narayanan, A. and Shmatikov, V. (2008). Robust deanonymization of large sparse datasets. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 111–125. IEEE.

[Nguyen, 2015] Nguyen, L. (2015). Introduction to a framework of e-commercial recommendation algorithms. *American Journal of Computer Science and Information Engineering*, 2(4):33–44.

[Oard et al., 1998] Oard, D. W., Kim, J., et al. (1998). Implicit feedback for recommender systems. In *Proceedings of the AAAI workshop on recommender systems*, pages 81–83.

[Ortega et al., 2013] Ortega, F., SáNchez, J.-L., Bobadilla, J., and GutiéRrez, A. (2013). Improving collaborative filtering-based recommender systems results using pareto dominance. *Information Sciences*, 239:50–61.

[Owen et al., 2011] Owen, S., Anil, R., Dunning, T., and Friedman, E. (2011). *Mahout in Action*. Manning Publications Co., Manning Publications Co. 20 Baldwin Road PO Box 261 Shelter Island, NY 11964, first edition.

[Pagare and Shinde, 2013] Pagare, R. and Shinde, A. (2013). Recommendation system using bloom filter in mapreduce. *International Journal of Data Mining & Knowledge Management Process*, 3(6):127.

[Pan et al., 2010] Pan, P.-Y., Wang, C.-H., Horng, G.-J., and Cheng, S.-T. (2010). The development of an ontology-based adaptive personalized recommender system. In *Electronics and Information Engineering (ICEIE), 2010 International Conference On*, volume 1, pages V1–76. IEEE.

[Pazzani, 1999] Pazzani, M. J. (1999). A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, 13(5-6):393–408.

[Peis et al., 2008] Peis, E., del Castillo, J. M., and Delgado-López, J. (2008). Semantic recommender systems. analysis of the state of the topic. *Hipertext. net*, 6:1–5.

[Pilászy and Tikk, 2009] Pilászy, I. and Tikk, D. (2009). Recommending new movies: even a few ratings are more valuable than metadata. In *Proceedings of the third ACM conference on Recommender systems*, pages 93–100. ACM.

[Pilászy et al., 2010] Pilászy, I., Zibriczky, D., and Tikk, D. (2010). Fast als-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 71–78. ACM.

[Rashid et al., 2002] Rashid, A. M., Albert, I., Cosley, D., Lam, S. K., McNee, S. M., Konstan, J. A., and Riedl, J. (2002). Getting to know you: learning new user preferences in recommender systems. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 127–134. ACM.

[Rashid et al., 2008] Rashid, A. M., Karypis, G., and Riedl, J. (2008). Learning preferences of new users in recommender systems: an information theoretic approach. *ACM SIGKDD Explorations Newsletter*, 10(2):90–100.

[Rendle and Schmidt-Thieme, 2008] Rendle, S. and Schmidt-Thieme, L. (2008). Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 251–258. ACM.

[Rubens et al., 2011] Rubens, N., Kaplan, D., and Sugiyama, M. (2011). Active learning in recommender systems. In *Recommender Systems Handbook*, pages 735–767. Springer.

[Rubens and Sugiyama, 2007] Rubens, N. and Sugiyama, M. (2007). Influence-based collaborative active learning. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 145–148. ACM.

[Said and Bellogín, 2014] Said, A. and Bellogín, A. (2014). Comparative recommender system evaluation: benchmarking recommendation frameworks. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 129–136. ACM.

[Salakhutdinov et al., 2007] Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798. ACM.

[Sarwar et al., 2001] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM.

[Sarwar et al., 2002] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2002). Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Fifth International Conference on Computer and Information Science*, pages 27–28. Citeseer.

[Sarwat et al., 2014] Sarwat, M., Levandoski, J. J., Eldawy, A., and Mokbel, M. F. (2014). Lars*: An efficient and scalable location-aware recommender system. *IEEE Transactions on Knowledge and Data Engineering*, 26(6):1384–1399.

[Schafer et al., 2007a] Schafer, J. B., Frankowski, D., Herlocker, J., and Sen, S. (2007a). Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer.

[Schafer et al., 2007b] Schafer, J. B., Frankowski, D., Herlocker, J., and Sen, S. (2007b). Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer.

[Shambour and Lu, 2011] Shambour, Q. and Lu, J. (2011). A hybrid multi-criteria semantic-enhanced collaborative filtering approach for personalized recommendations. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2011 IEEE/WIC/ACM International Conference on*, volume 1, pages 71–78. IEEE.

[Shani and Gunawardana, 2011] Shani, G. and Gunawardana, A. (2011). Evaluating recommendation systems. In *Recommender systems handbook*, pages 257–297. Springer.

[Shannon, 2001] Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55.

[Sobhanam and Mariappan, 2013] Sobhanam, H. and Mariappan, A. (2013). Addressing cold start problem in recommender systems using association rules and clustering technique. In *Computer Communication and Informatics (ICCCI), 2013 International Conference on*, pages 1–5. IEEE.

[Su and Khoshgoftaar, 2009] Su, X. and Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4.

[Swamidass and Baldi, 2007] Swamidass, S. J. and Baldi, P. (2007). Mathematical correction for fingerprint similarity measures to improve chemical retrieval. *Journal of chemical information and modeling*, 47(3):952–964.

[Takács et al., 2007] Takács, G., Pilászy, I., Németh, B., and Tikk, D. (2007). Major components of the gravity recommendation system. *ACM SIGKDD Explorations Newsletter*, 9(2):80–83.

[Takács and Tikk, 2012] Takács, G. and Tikk, D. (2012). Alternating least squares for personalized ranking. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 83–90. ACM.

[Tingle et al., 2010] Tingle, D., Kim, Y. E., and Turnbull, D. (2010). Exploring automatic music annotation with acoustically-objective tags. In *Proceedings of the international conference on Multimedia information retrieval*, pages 55–62. ACM.

[Tintarev and Masthoff, 2011] Tintarev, N. and Masthoff, J. (2011). Designing and evaluating explanations for recommender systems. In *Recommender Systems Handbook*, pages 479–510. Springer.

[Tiroshi et al., 2012] Tiroshi, A., Kuflik, T., Kay, J., and Kummerfeld, B. (2012). Recommender systems and the social web. *Advances in User Modeling*, pages 60–70.

[Trewin, 2000] Trewin, S. (2000). Knowledge-based recommender systems. *Encyclopedia of library and information science*, 69(Supplement 32):180.

[Tso-Sutter et al., 2008] Tso-Sutter, K. H., Marinho, L. B., and Schmidt-Thieme, L. (2008). Tag-aware recommender systems by fusion of collaborative filtering algorithms. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1995–1999. ACM.

[Uchyigit, 2009] Uchyigit, G. (2009). Semantically enhanced web personalization. In *Web Mining Applications in E-commerce and E-services*, pages 25–44. Springer.

[van den Oord et al., 2013] van den Oord, A., Dieleman, S., and Schrauwen, B. (2013). Deep content-based music recommendation. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 2643–2651. Curran Associates, Inc.

[Vogoo, 2013] Vogoo (2013). Vogoo.

[Voß et al., 2009] Voß, A., Newell, C., and Marrow, P. (2009). Mymedia: Dynamic personalization of multimedia. *Proceedings of EuroITV09, Leuven, Belgium*.

[Vozalis and Margaritis, 2007] Vozalis, M. G. and Margaritis, K. G. (2007). Using svd and demographic data for the enhancement of generalized collaborative filtering. *Information Sciences*, 177(15):3017–3037.

[Wang et al., 2006] Wang, J., De Vries, A. P., and Reinders, M. J. (2006). Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 501–508. ACM.

[Weng et al., 2008] Weng, L.-T., Xu, Y., Li, Y., and Nayak, R. (2008). Exploiting item taxonomy for solving cold-start problem in recommendation making. In *2008 20th IEEE International Conference on Tools with Artificial Intelligence*, volume 2, pages 113–120. IEEE.

[Werner et al., 2013] Werner, D., Cruz, C., and Nicolle, C. (2013). Ontology-based recommender system of economic articles. *arXiv preprint arXiv:1301.4781*.

[Xia et al., 2006] Xia, Z., Dong, Y., and Xing, G. (2006). Support vector machines for collaborative filtering. In *Proceedings of the 44th annual Southeast regional conference*, pages 169–174. ACM.

[Xu et al., 2012] Xu, B., Bu, J., Chen, C., and Cai, D. (2012). An exploration of improving collaborative recommender systems via user-item subgroups. In *Proceedings of the 21st international conference on World Wide Web*, pages 21–30. ACM.

[Zhang et al., 2011] Zhang, Z.-K., Zhou, T., and Zhang, Y.-C. (2011). Tag-aware recommender systems: a state-of-the-art survey. *Journal of computer science and technology*, 26(5):767–777.

[Zheng et al., 2008] Zheng, R., Wilkinson, D., and Provost, F. (2008). Social network collaborative filtering. *Stern, IOMS Department, CeDER, Vol*.

[Zheng et al., 2015] Zheng, Y., Mobasher, B., and Burke, R. (2015). Carskit: A java-based context-aware recommendation engine. In *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, pages 1668–1671. IEEE.

[Zhou et al., 2011] Zhou, K., Yang, S.-H., and Zha, H. (2011). Functional matrix factorizations for cold-start recommendation. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 315–324. ACM.

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

[Zhou et al., 2008] Zhou, Y., Wilkinson, D., Schreiber, R., and Pan, R. (2008). Large-scale parallel collaborative filtering for the netflix prize. In *Algorithmic Aspects in Information and Management*, pages 337–348. Springer.

[Zhuo et al., 2011] Zhuo, G., Sun, J., and Yu, X. (2011). A framework for multi-type recommendations. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on*, volume 3, pages 1884–1887. IEEE.

[Ziegler et al., 2004] Ziegler, C.-N., Lausen, G., and Schmidt-Thieme, L. (2004). Taxonomy-driven computation of product recommendations. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 406–415. ACM.

[Ziegler et al., 2005] Ziegler, C.-N., McNee, S. M., Konstan, J. A., and Lausen, G. (2005). Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, pages 22–32. ACM.

Annexes

Appendix A

Evaluating the performance of Recommender Systems

A.1 Introduction

Recommender systems answer to different challenges depending on the context and on the goal. For instance, recommendations may need to be suited in terms of attributes, e.g. to avoid user's allergies in recipes recommendations. As a consequence, there are many method and techniques to evaluate the performance of these systems. In general, one can sum up these techniques into: prediction-oriented, raking-oriented and content-oriented metrics. For further explanations about evaluation metrics in recommender systems please refer to [Shani and Gunawardana, 2011]

In order to evaluate a recommender system, the most common approach is to split the current dataset into training set and test set[1]. The main idea is to train the recommendation technique using the ratings of users (u) in items (i) within the training set, i.e. $r_{u,i}$. The technique identifies users, items and rating patterns following the recommendation technique assumption, and hence it is possible to predict the users' ratings in items ($\hat{r}_{u,i}$)) which are absent in the training set. On the other hand, the test set contains (normally[2]) other ratings from same users and items that appear in the training set. As a consequence, by comparing the predictions computed for users and items to the ratings in test set, it is

[1] Further dataset's split are possible and sometimes required to validate algorithm's parameters and hyper-parameters.

[2] Odd cases and exceptions may appear when users' and/or items' ratings is low.

Table A.1: Notation used in the evaluation metrics for recommender systems.

Notation	Description
R	Set of ratings
$\|R\|$	Number of ratings
U	Set of users
$\|U\|$	Number of users
u	particular user
I	Set of items
$\|I\|$	Number of items
i	particular item
$r_{u,*}$	Set of user's ratings
$r_{*,i}$	Set of item's ratings
$r_{u,i}$	Rating of user u in item i
$\hat{r}_{u,i})$	Predicted rating of user u in item i
$\hat{r}_{u,i})$	Predicted rating of user u in item i
r_{max}	Maximum possible rating value
r_{min}	Maximum possible rating value
$r_{threshold}$	Rating threshold to assume the relevancy of an item for a user, $rel_{u,i}$
$top-K$	Set of K items to recommend

possible evaluate the performance and quality of the recommendations.

This appendix aims to cover most of the evaluation methods used in the recommender systems field. Table A.1 shows the notation used in this appendix.

A.2 Prediction oriented evaluations

This evaluation is the most widely used in literature [Kantor et al., 2011]. Recommender systems predict the interest of users in items, $\hat{r}_{u,i})$, which are the deduction of how a user would rate an item. This allows to aim to particular user and item. Prediction oriented approaches aim to evaluate the closeness between these predictions and real observed values. Thus, they use the training set to train a get recommendations and the test set to compare if recommendations were correct, regarding their predicted ratings and the real observed ratings from the test set.

A.2.1 MAE: Mean Absolute Error

This method measures the absolute difference between predictions ($\hat{r}_{u,i}$)) and observations ($r_{u,i}$): $e_{u,i} = |r_{u,i} - \hat{r}_{u,i}|$. As a consequence, the lower is this error, the better is the prediction. The measure is computed for the all the ratings in the test set; thus it represents the average of the absolute difference.

$$MAE = \frac{1}{|R|} * \sum_{r_{u,i} \subset R} |r_{u,i} - \hat{r}_{u,i}| = \frac{1}{|R|} * \sum_{r_{u,i} \subset R} |e_{u,i}| \tag{A.1}$$

Other variances propose to compute the squared root of MAE. In addition, it is possible to define a Normalized Mean Absolute Error (NMAE) as the MAE divided by the range of the ratings values.

$$NMAE = \frac{MAE}{r_{max} - r_{min}} \tag{A.2}$$

Finally, datasets may contain users or items that appear more frequently. This affects the distribution of users and items in training and test sets. It is possible to compute and normalize the MAE depending on users or items (instead of the overall number of ratings). In fact, one may compute a user oriented metric (MAE_u) or an item oriented metric (MAE_i), and then normalize by the number of users or items that contributed to this metric.

$$MAE_u = \frac{1}{|R_u|} * \sum_{r_{u,i} \subset R_u} |r_{u,i} - \hat{r}_{u,i}| \tag{A.3}$$

$$U - MAE = \frac{1}{|U|} * \sum_{u \subset U} |MAE_u| \tag{A.4}$$

$$MAE_i = \frac{1}{|R_i|} * \sum_{r_{*,i} \subset R_i} |r_{u,i} - \hat{r}_{u,i}| \tag{A.5}$$

$$I - MAE = \frac{1}{|I|} * \sum_{i \subset I} |MAE_i| \tag{A.6}$$

A.2.2 RMSE: Root Mean Square Error

This method measures the squared difference between predictions ($\hat{r}_{u,i}$)) and observations ($r_{u,i}$): $e_{u,i} = |r_{u,i} - \hat{r}_{u,i}|$. In particular, it represents the standard deviation of this difference, thus it is also dubbed Root Mean Square Deviation (RMSD). Compared to MAE, this technique provokes higher penalizations when both values are distant. As a result, the lower is this error, the better is the prediction. The measure is computed for all the ratings in the test set; thus it represents the average of the squared difference.

$$RMSE = \sqrt{\frac{1}{|R|} * \sum_{r_{u,i} \subset R} (r_{u,i} - \hat{r}_{u,i})^2} = \sqrt{\frac{1}{|R|} * \sum_{r_{u,i} \subset R} e_{u,i}^2} \tag{A.7}$$

Normalized Root Mean Square Error (NRSME) is the RMSE divided by the range of the variable.

$$NRMSE = \frac{RMSE}{r_{max} - r_{min}} \tag{A.8}$$

Finally, datasets may contain users or items that appear more frequently. This affects the distribution of users and items in training and test sets. It is possible to compute and normalize the MAE depending on users or items (instead of the overall number of ratings). In fact, one may compute a user oriented metric ($RMSE_u$) or an item oriented metric ($RMSE_i$), and then normalize by the number of users or items that contributed to this metric.

$$RMSE_u = \sqrt{\frac{1}{|R_u|} * \sum_{r_{u,i} \subset R_u} (r_{u,i} - \hat{r}_{u,i})^2} \tag{A.9}$$

$$U - RMSE = \frac{1}{|U|} * \sum_{u \subset U} |RMSE_u| \tag{A.10}$$

$$RMSE_i = \sqrt{\frac{1}{|R_i|} * \sum_{r_{*,i} \subset R_i} (r_{u,i} - \hat{r}_{u,i})^2} \tag{A.11}$$

$$I - RMSE = \frac{1}{|I|} * \sum_{i \subset I} |RMSE_i| \tag{A.12}$$

A.3 Ranking oriented evaluations

Ranking evaluation are related to usefulness of recommendations. They do not predict singular user item ratings, but try to collect and present a set of items of the user's interest. This set of items are likely to be "consumed" (e.g. visited, watched, etc) by the user in a close future.

The presence of items in this set depends on three factors: the rating prediction value ($\hat{r}_{u,i}$), the size of the set (K), a.k.a. top-K, and the relevancy threshold ($r_{threshold}$). The predictions are inherit to the model. The size of the set is a trade-off, short values make short list of items to choose over whether large values make more difficult to the user the decision makings. The relevancy threshold sets whether an item is relevant or not for a given user, and thus whether the item should be or not inserted in the list.

As a result, the $rel_{u,i}$ is a binary value to represent whether an item i is relevant or not for a user u. It is possible compare the item's prediction with user's ratings in the test set. If the real observed rating in the test is equal or above $r_{threshold}$, the item i is relevant for the user, and $rel_{u,i} = 1$, otherwise $rel_{u,i} = 0$.

A.3.1 Precision, Recall and F-Measures

Precision and Recall evaluates the relevancy of recommendations in terms of probabilities, regarding the top-K items or the users' ratings.

Precision is the probability that an item i in top-K is relevant for the user u over all other items in top-K. The higher the probability is, the better is the precision. An overall value of 1.0 means that every item recommended in top-K is relevant. Yet, it does not say that all relevant information have been suggested.

$$Precision_u@K = \frac{1}{K} * \sum_{i \subset K} rel_{u,i} \tag{A.13}$$

The overall precision of the system is given by the users' precision average.

$$Precision@K = \frac{1}{|U|} * Precision_u@K \tag{A.14}$$

Recall is the probability that an item i in top-K is relevant for the user u over all other users' items ratings. The average value of 1.0 means all the good recommendations are in the top-K, and thus the higher, the better.

$$Recall_u@K = \frac{1}{R_u} * \sum_{i=1}^{K} rel_{u,i} \tag{A.15}$$

The overall recall of the system is given by the users' recall average.

$$Recall@K = \frac{1}{|U|} * Recall_u@K \tag{A.16}$$

The F-β-measure refers also to the top-K elements. It is the Harmonic Mean of Precision and Recall and it might balance or weight precision and recall values. The weight value is denoted as β. Typically, the F-Measure uses $\beta = 1$ in order to equally balance precision and recall.

$$F - \beta measure@K = (1 + \beta^2) * \frac{Precision@K * Recall@K}{\beta^2 * Precision@K + Recall@K} \tag{A.17}$$

A.3.2 DCG: Discounted Cumulative Gain

Discounted Cumulative Gain (DCG) measures also the relevancy of items [Järvelin and Kekäläinen, 2002]. This measure takes into account the position of items. It is based on two assumptions: (1) most relevant items should appear higher in the user's top K recommendations since they are more useful for the user, and (2) the relevancy is a marginal factor for irrelevant items. In this measure, the higher, the better. We calculate the overall value for all users U.

$$DCG_u@K = \sum_{i=1}^{K} \frac{rel_{u,i}}{log_2(1 + i)} \tag{A.18}$$

$$DCG@K = \frac{1}{|U|} * DCG_u@K \tag{A.19}$$

Finally, the Normalized Discounted Cumulative Gain (NDCG) is the current status of $DCG@K$ compared to the $IdealDCG@K$.

$$NDCG@K = \frac{DCG@K}{IdealDCG@K} \tag{A.20}$$

A.4 Content oriented evaluations

The recommender systems use to suggest users a set of items (top-K) that best suits their interests. Depending on the domain of recommendation, it is interesting to balance the degree of similarity of the items which are recommended to the user. For instance, to recommend very similar movies to one user might bore her. On the contrary, to recommend very different movies may confuse her and give the impression of bad recommendations. This trade-off is called similarity/diversity [Su and Khoshgoftaar, 2009].

In addition, users may appreciate as well that recommendation are renew even if they are not consumed. Novelty metrics show the difference between the information presented and the information that usually is presented [Castells et al., 2011].

A.4.1 ILS: Intra-List Similarity

It is also called ILD (Intra-List Diversity). It measures the diversity/similarity of items in top-K [Ziegler et al., 2005]. The similarity $sim(i, j)$ between two items i, j depends on the characteristics they share. For instance, two movies of the same genre or belonging to the same saga are considered to be similar. The higher the value is, the more similar the items in the top are.

$$ILS_u@K = \frac{1}{2} * \sum_{i \subset K} \sum_{j \subset K} sim(i, j) \tag{A.21}$$

The global similarity for the system is given by the users' average ILS:

$$ILS@K = \frac{1}{|U|} * ILS_u@K \tag{A.22}$$

A.4.2 EBN: Entropy-Based Novelty

It measures the quantity of information of K presented items [Bellogín et al., 2010]. This technique sees items as variables with associated probabilities (entropy). We might consider that items have an equal probability of appearance $p_i = 1/K$. Hence, the combination C_i of all probabilities generates the global entropy. The higher, the better.

$$EBN_u@k = C_i = log_2(k) = log_2(\frac{1}{1/K}) = log_2(\frac{1}{p}) = -log_2(p) \tag{A.23}$$

When the probabilities are not equal, the entropy is weighted by these probabilities.

$$EBN_u@k = -p_1 * log_2(p_1) - p_2 * log_2(p_2) - ... - p_K * log_2(p_K) = -\sum_{i \subset K} p_i * log_2(p_i) \tag{A.24}$$

Thus, the global EBN for the system is given by the users' average:

$$EBN@K = \frac{1}{|U|} * EBN_u@K \tag{A.25}$$

A.4.3 MSI: Normalized Mean Self-Information

This metric evaluates the information novelty between predictions in the top K and known user's preferences R_u [Zhou et al., 2010]. The higher, the better.

$$MSI_u@K = \frac{1}{|K|} * \sum_{i \subset K} log(\frac{|U|}{|(v \subset U | i \subset R_u)|}) \tag{A.26}$$

Thus, the global EBN for the system is given by the users' average:

$$MSI@K = \frac{1}{|U|} * MSI_u@K \tag{A.27}$$

Appendix B

Bloom Filters

B.1 Introduction

The concept of bloom filter was first introduced by Burton H. Bloom in 1970 [Bloom, 1970]. It has been used in many different applications, such as databases queries or computer networks, due to its memory-efficiency and fast-capabilities [Broder and Mitzenmacher, 2004]. To the best of our knowledge, this structure has not been used yet in the field of recommender systems.

B.2 Definition and properties of Bloom Filters

A bloom filter is a bit-structure, which represents "n"-elements of the same set "S" in a lower space of "m"-bits [Broder and Mitzenmacher, 2004]. Initially, the m-bits are set to "0" representing the absence of elements in the filter. Then, "k"-independent hash functions efficiently distribute the insertion of elements among the bit-structure. This modifies the status of "k" bits in the structure (one bit for each hash function used). In other words, to insert an element it is hashed to get the "k"-positions of the structure to set to "1".

Bloom filter structure allows fast membership queries. It works as follows: the requested element is hashed with the "k" hash functions and the "k" resulted indexes are checked in the bit-structure. If any of these indexes is set to "0", the bloom filter *assures that the element has not been inserted.* When all indexes are set to "1", one can assume that

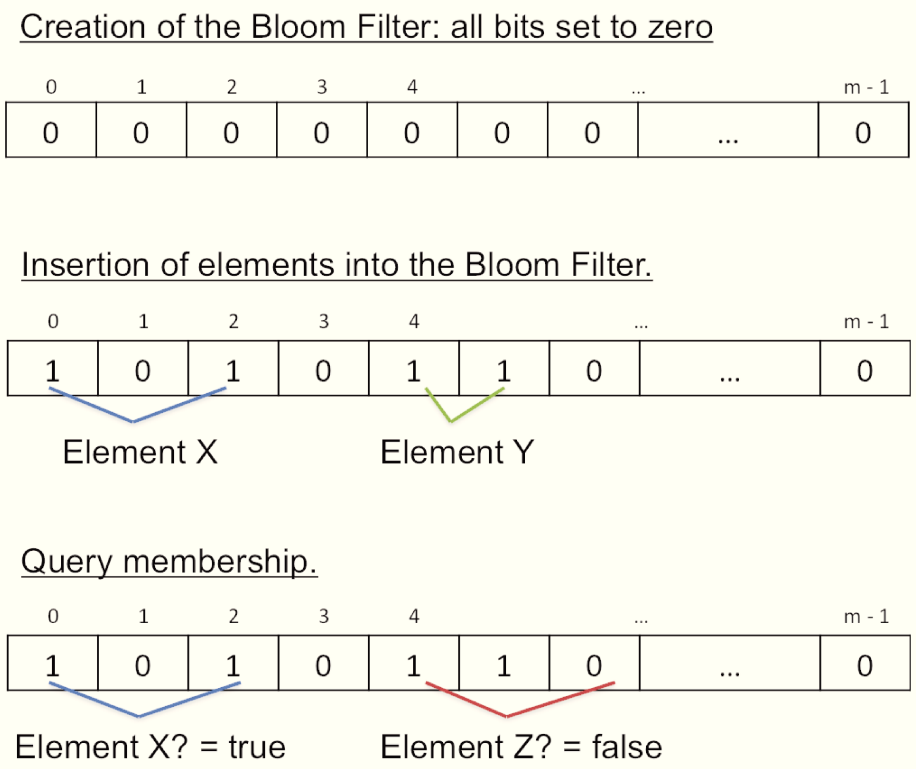


Figure B.1: Example of bloom filter. Initially the bloom filter is empty (bits set to "0"). Elements X, Y are inserted by hashing them and by setting adequate bits to "1". One membership query for element X returns true, while for a non-inserted element Z returns false.

the element has been inserted with an error probability called false positive probability "fp". A false positive is the situation in which a membership query returns that the element belongs to the set, yet it actually does not. Nevertheless, the estimation of this false positive ratio is possible and near-optimal parameters may be computed [Broder and Mitzenmacher, 2004]. A bloom filter example is shown in Figure B.1.

In addition, the intersection and the union of two filters, A and B, are possible under two conditions: (1) the bits number of A is equal to the bits number of B, and (2) they both use the same hash functions. i.e m and k are equal in both bloom filters. These properties will allow us to perform filter comparisons and to develop our approach.

B.2.1 Near Optimal False Positive

As explained above, one may assure that an element has not been inserted in the filter if any of the associated indexes is set to "0". On the other hand, the probability of a false positive can be estimated. Indeed, the false positive ratio depends on three parameters: (1) the quantity of expected insertions "n", (2) the size of the bit structure "m", and (3) the number of hash functions "k". After all the "n" elements have been inserted, some bits in the structure remain set to "0" but others are set to "1". The *probability that a specific bit is still zero* is given by Equation B.1 [Broder and Mitzenmacher, 2004].

$$p(bit = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx \left(e^{-kn/m}\right) \tag{B.1}$$

Moreover, the *probability that a specific bit is set to one* is given by $q(bit = 1) = 1 - p(bit = 0)$. Thus, the false positive probability is defined through this probability and the number of hash functions[1], as in Equation B.2.

$$fp = q(bit = 1)^k = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k \tag{B.2}$$

Further efforts to find out near-optimal values for the false positive ratio can demonstrate the existence of a global minimum [Broder and Mitzenmacher, 2004] for "k", $k = \frac{m \cdot ln2}{n}$. In addition, "$m$" can be estimated by: $m \geq \frac{-n \cdot log_2(fp)}{ln2}$. Note that in a real application "k", "m" and "n" are integer numbers. These equations are used to build and initialize any bloom filter.

B.2.2 Intersection property

The intersection of two bloom filters, BF_A and BF_A, aims to find common elements of two different sets, S_A and S_B. One bit will be set to one if: (1) the element belongs to the intersection of BF_A and BF_B: $(BF_A \cap BF_B)$, or (2) this bit is set simultaneously to one in $BF_A - BF_A \cap BF_B$ and $BF_B - BF_A \cap BF_B$. Indeed, it represents the **AND operation** of the two filters A and B, and the result is a new bloom filter containing their common

[1] Assuming perfectly random hash functions

insertions. Thus, the *probability that a specific bit is set to one in both filters* is given by the simplified Equation 5.1 [Broder and Mitzenmacher, 2004] and the false positive of this intersection is given by $fp = q(bit = 1)^k$.

$$q(bit = 1) = 1 - (1 - \frac{1}{m})^{k \cdot |S_A|} - (1 - \frac{1}{m})^{k \cdot |S_B|} + (1 - \frac{1}{m})^{k \cdot (|S_A| + |S_B| - |S_A \cap S_B|)} \quad \text{(B.3)}$$

B.2.3 Union property

The union of two bloom filters, BF_A and BF_A, aims to join two different sets, S_A and S_B. This is the **OR operation** of these filters and results in a new bloom filter joining both filters. Therefore, the resulted filter union represents a new set $S_A \cup S_B$. It is possible to approximate the size of the expected total insertion in a bloom filter [Swamidass and Baldi, 2007]. Equation B.4 shows this approximation, where $card(X)$ denotes the number of bits set to "1" in the bloom filter "X". By considering both Equations B.4 and B.2, it is possible to know the false positive ratio of the union of two bloom filters.

$$n^* = -\frac{m \cdot ln\Big(1 - \frac{card(X)}{m}\Big)}{k} \quad \text{(B.4)}$$

B.3 Types of Bloom Filters

In this section we have explained a very simple case of representing items based on the standard definition of bloom filters, where we have supposed a static dataset. In fact, when new features need to be taken into consideration the number of insertions raises. However, the current bloom filter parameters (size "m" and hash functions "k") can not face such raise and yield to a false positive raise as well. Indeed, these parameters cannot be modified, hence when new features are added to the system all bloom filters need to be re-built according to new parameters. In addition, standard bloom filters do not count the number of insertions of the same element, if necessary. Thus, these filters can not be compared to frequency vectors.

In fact, these situations correspond to static versus dynamic datasets and binary versus

frequency vector representations. In our model we use standard bloom filters for simplicity and explanation, yet these drawbacks can be addressed by other bloom filter approaches. On the one hand, dynamic datasets are possible to model by using dynamic/scalable bloom filters [Guo et al., 2010, Almeida et al., 2007]. The main idea is that a scalable bloom filter is formed by one or more bloom filters. Thus, these filters are built by blocks, and hence by adding new blocks to the filter one may add a new set of features. As a consequence, this avoid to re-build the bloom model when new features are added. In addition, similar probability inductions given in this section can be applied to scalable and dynamic probabilities, and thus our bloom model remains possible in these cases. On the other hand, counting insertions is possible as well by using counting bloom filters [Broder and Mitzenmacher, 2004]. These filters contains new bit-sets to count the insertions performed. Thus, if one insertion has been made several times, filters may approximate the counting. As a result, one may compare counting filters to frequency vectors, with more complexity. Other variances of bloom filters deal with extra compression[Broder and Mitzenmacher, 2004].

Glossaire

- ALS : Alternating Least Squares
- BF : Bloom Filter
- CB : Content-Based
- CBF : Counting Bloom Filter
- CF : Collaborative Filtering
- $DSGD$: Distributed Stochastic Gradient Descent
- EBN : Entropy-Based Novelty
- FP : False Positive
- FN : False Negative
- GD : Gradient Descent
- $HELF$: Harmonic Entropy Logarithmic Frequency
- ILD : Intra-List Diversity
- ILS : Intra-List Similarity
- IR : Information Search and Retrieval
- KNN : K-Nearest Neighbours
- LDA : Latent Dirichlet Allocation
- LPE : Logarithmic Popularity Entropy

- MAE : Mean Absolute Error
- MF : Matrix Factorization
- NLP : Natural Language Processing
- PCA : Principal Component Analysis
- RBM : Restricted Boltzmann Machines
- $RMSE$: Root Mean Square Error
- RS : Recommender Systems
- SGD : Stochastic Gradient Descent
- SVD : Singular Value Decomposition
- SVM : Support Vector Machine
- $TF - IDF$: Term Frequency-Inverse Document Frequency
- TP : True Positive
- TN : True Negative

Manuel Pozo

TOWARDS ACCURATE AND SCALABLE RECOMMENDER SYSTEMS

Résumé :
Les systèmes de recommandation permettent de filtrer et présenter en premier les informations susceptibles d'intéresser les utilisateurs. Ce domaine a suscité l'intérêt du e-commerce afin de prédire les préférences des internautes et personnaliser les offres et produits qui leur sont proposés. Ces systèmes font face à de grands défis: ils souffrent du démarrage à froid, ils exigent un traitement de grand volume de données qui peuvent être très hétérogènes. Une des techniques populaire est la factorisation matricielle qui a démontré une précision dans les prédictions et un bon passage à l'échelle.
Nos contributions se concentrent sur 4 aspects: (1) un meilleur passage à l'échelle de la factorisation matricielle, (2) l'intérêt implicite des utilisateurs dans les attributs des articles, (3) la représentation très compacte des caractéristiques des utilisateurs/items, et (4) l'apprentissage actif pour faire face au démarrage à froid. Nos expérimentations en utilisant des jeux de données publics démontrent les performances de nos approches.
Mots clés : filtrage collaboratif, système de recommandation, distribution, filtre de bloom, demarrage à froid, apprentissage actif

Abstract :
Recommender Systems filter and present first the information in which users may be interested. This has raised the attention of the e-commerce to predict future interests and to personalize the offers (a.k.a. items). These systems face great challenges: they require distributed techniques to deal with a huge volume of data, they aim to exploit very heterogeneous data, and they suffer from cold-start. Among popular techniques, Matrix Factorization has demonstrated high accurate predictions and scalability.
Our contributions are given by four aspects: (1) the distribution of a matrix factorization algorithm, (2) the implicit interest of the users in the attributes of the items to be used by matrix factorization, (3) the representation of users/items through a high quantity of features in low memory-consumption, and (4) the active learning techniques to cope with cold-start. We demonstrate their performance in terms of accuracy and efficiency using a publicly available dataset.
Keywords : collaborative filtering, recommender system, distribution, bloom filter, cold-start, active learning