Local and social recommendation in decentralized architectures

Simon Meyffret

To cite this version:

Simon Meyffret. Local and social recommendation in decentralized architectures. Other [cs.OH]. INSA de Lyon, 2012. English. NNT : 2012ISAL0127 . tel-00833220

HAL Id: tel-00833220
https://theses.hal.science/tel-00833220

Submitted on 12 Jun 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° D'ORDRE ANNÉE 2012

THÈSE

LOCAL AND SOCIAL RECOMMENDATION IN DECENTRALIZED ARCHITECTURES

RECOMMANDATION LOCALE ET SOCIALE DANS LES ARCHITECTURES DÉCENTRALISÉES

Présentée devant :
L'Institut National des Sciences Appliquées de Lyon
Pour obtenir :
Le grade de docteur
Spécialité :
Informatique
Formation doctorale :
Informatique
École doctorale :
Informatique et Mathématiques
Par :
Simon MEYFFRET

SOUTENUE PUBLIQUEMENT LE 7 DÉCEMBRE 2012 DEVANT LE JURY COMPOSÉ DE :

Catherine BERRUT, Professeur des Universités, Polytech Grenoble	Examinatrice
Sylvie CALABRETTO, Professeur des Universités, INSA Lyon	Examinatrice
Max CHEVALIER, Maître de Conférences HDR, Université Paul Sabatier	Rapporteur
Frédérique LAFOREST, Professeur des Universités, Telecom Saint-Etienne	Directrice de thèse
Lionel MÉDINI, Maître de Conférences, Université Lyon 1	Co-directeur de thèse
Daniele QUERCIA, Docteur, Yahoo ! Labs	Examinateur
François TAÏANI, Professeur des Universités, Université de Rennes 1	Rapporteur

LABORATOIRE D'INFORMATIQUE EN IMAGE ET SYSTÈMES D'INFORMATION

Success is the ability to go from failure to failure without losing your enthusiasm.
— Winston Churchill

REMERCIEMENTS

Cette thèse est le fruit de trois années de travail en collaboration avec Frédérique Laforest et Lionel Médini. Je les remercie affectueusement, ils ont su m'apporter bien plus qu'un cadre scientifique quand j'en avais besoin.

Je tiens à remercier mes rapporteurs – Max Chevalier et François Taïani – ainsi que mes relecteurs – Stéphane Frénot, Christophe Gravier et Julien Subercaze – pour leurs remarques constructives et pertinentes.

Merci au professeur Boualem Benatallah pour son accueil chaleureux en Australie, ainsi qu'au LIRIS, à l'ED et à la région Rhône Alpes pour leur financement. Ce séjour m'a enrichi sur un plan professionnel et personnel. J'y ai rencontré des gens qui me sont aujourd'hui très proches.

Je remercie les membres de mon équipe de recherche DRIM qui ont toujours su répondre à mes questions avec beaucoup d'efficacité, et le personnel administratif du LIRIS, sans qui, il faut bien l'avouer, j'aurais été perdu.

Ce doctorat n'aurait pas eu la même saveur sans mes collègues et amis Arnaud et Benjamin, qui ont eu l'obligeance de m'apporter des points de vues très intéressants et m'ont offert des discussions passionnées et passionnantes. Je remercie Deming, Sabina et Usman pour leur patience, ce fut un plaisir de partager votre bureau.

Je tiens à souligner la bonne humeur et l'optimisme (presque) toujours requinquants d'Elise. Sans oublier Brice et Pierre-Nico, partenaires de pause café, et Julien, sans qui ma première année de doctorat aurait été bien morne.

Une pensée sincère pour Julie, qui m'a soutenu dans mes débuts difficiles et qui a su avec beaucoup d'ingéniosité et de persévérance détourner mes pensées négatives lors de la rédaction de ce manuscrit.

Je n'oublie pas ma famille, un soutien sans faille, et mes amis, toujours là pour m'aider à décompresser.

Enfin je remercie mes éventuels lecteurs pour leur motivation !

ABSTRACT

Recommender systems are widely used to achieve a constantly growing variety of services. Alongside with social networks, recommender systems that take into account friendship or trust between users have emerged.

In this thesis, we propose an evolution of trust-based recommender systems adapted to decentralized architectures that can be deployed on top of existing social networks. Users profiles are stored locally and are exchanged with a limited, user-defined, list of trusted users.

Our approach takes into account friends' similarity and propagates recommendation to direct friends in the social network in order to prevent ratings from being globally known. Moreover, the computational complexity is reduced since calculations are performed on a limited dataset, restricted to the user's neighborhood.

On top of this propagation, our approach investigates several aspects. Our system computes and returns to the final user a confidence on the recommendation. It allows the user to tune his/her choice from the recommended products. Confidence takes into account friends' recommendations variance, their number, similarity and freshness of the recommendations.

We also propose several heuristics that take into account peer-to-peer constraints, especially regarding network flooding. We show that those heuristics decrease network resources consumption without sacrificing accuracy and coverage.

We propose default scoring strategies that are compatible with our constraints.

We have implemented and compared our approach with existing ones, using multiple datasets, such as Epinions and Flixster. We show that local information with default scoring strategies are sufficient to cover more users than classical collaborative filtering and trust-based recommender systems. Regarding accuracy, our approach performs better than others, especially for cold start users, even if using less information.

RÉSUMÉ

Dans notre société de plus en plus numérique, les systèmes de recommandation ont fait leur apparition dans le but de résoudre le problème bien connu de surcharge d'information. L'adoption des réseaux sociaux a permis l'émergence de systèmes intégrant les relations sociales dans leurs recommandations.

Dans cette thèse, nous proposons un système de recommandation adapté aux architectures décentralisées pouvant être déployé sur des réseaux sociaux existants. L'utilisateur conserve son profil en local et ne communique qu'avec un ensemble restreint d'utilisateurs de confiance, avec qui il accepte de partager ses données.

Nous prenons en compte le réseau social de l'utilisateur afin de construire le réseau de pairs. La similarité des amis est prise en compte pour pondérer les liens. Les recommandations sont propagées dans le réseau, passant d'amis en amis jusqu'à atteindre l'utilisateur désiré. Ainsi seuls les amis directs communiquent entre eux.

À partir de cette propagation, nous proposons plusieurs techniques. Tout d'abord, nous délivrons à l'utilisateur final une confiance du système dans la fiabilité de la recommandation. Ceci lui permet de choisir parmi les produits sélectionnés, lesquels semblent effectivement les plus pertinents pour lui. Cette confiance est calculée sur plusieurs critères, tels que la variation des recommandations des amis, leur nombre, la similarité et la fraîcheur de la recommandation.

Ensuite, nous définissons des heuristiques adaptant notre approche aux systèmes pair-à-pair. Dans de telles architectures, le réseau est une ressource critique et ne doit pas être constamment surchargé. Ces heuristiques limitent la consommation réseau de notre approche tout en fournissant des recommandations pertinentes à l'utilisateur.

Enfin, nous proposons plusieurs stratégies de score par défaut, dans le cas où aucun score n'est calculable, prenant en compte les contraintes en terme d'accès à l'information par le système.

Nous comparons notre approche avec des approches classiques de recommandation, de filtrage collaboratif ou basées sur la confiance, en utilisant plusieurs jeux de données existants, tels qu'Epinions et Flixster, ainsi que deux jeux de données que nous avons construits nous-même. Nous montrons qu'une approche purement locale, associée à des stratégies de score par défaut, offre de meilleurs résultats que la plupart des autres approches, notamment en ce qui concerne les *cold start users*.

PUBLICATIONS

Some ideas and figures have appeared previously in the following publications:

[1] Simon Meyffret, Lionel Médini, and Frédérique Laforest. Trust-Based Local and Social Recommendation. In *RecSys 2012 Workshop on Recommender Systems and the Social Web*, pages 53–60, September 2012.

[2] Simon Meyffret, Lionel Médini, and Frédérique Laforest. Recommandation basée sur la confiance : une approche sociale et locale. *Document Numérique*, pages 33–56, August 2012.

[3] Simon Meyffret, Lionel Médini, and Frédérique Laforest. User-centric Trust-based Recommendation. In *International Conference on Information Technology-New Generations*, pages 707–713, April 2012.

[4] Simon Meyffret, Lionel Médini, and Frédérique Laforest. Trust-based recommendation with privacy. In *INFORSID11*, pages 369–384, May 2011.

CONTENTS

I INTRODUCTION 1

1 INTRODUCTION 3
  1.1 Context 3
  1.2 Motivations 4
  1.3 Contributions 5

II PREAMBLE 9

2 STATE OF THE ART ON RECOMMENDER SYSTEMS 11
  2.1 Content-based recommender systems 12
    2.1.1 Description of textual content 13
    2.1.2 Description of other kind of content 14
    2.1.3 User profile 14
    2.1.4 Conclusion 15
  2.2 Collaborative filtering recommender systems 16
    2.2.1 Centralized version 17
    2.2.2 Decentralized version 17
    2.2.3 Conclusion 18
  2.3 Trust-based recommender systems 18
    2.3.1 TidalTrust 20
    2.3.2 MoleTrust 21
    2.3.3 TrustWalker 21
    2.3.4 Conclusion 23
  2.4 Social-based recommender systems 23
    2.4.1 SocialMF 24
    2.4.2 Hoens et al. [HBC10] 25
    2.4.3 Conclusion 25
  2.5 Conclusion 26

3 DEFINITIONS / EXAMPLE 29
  3.1 Graph theory definitions 30
    3.1.1 Graphs 30
    3.1.2 Simple graphs 31
    3.1.3 Digraphs 32
    3.1.4 Bipartite graphs 32
    3.1.5 Weighted graphs 33
    3.1.6 Other graphs 34
  3.2 Vocabulary 35
    3.2.1 Social network 35
    3.2.2 Trust network 36
    3.2.3 Similarity network 36
    3.2.4 Ratings 37
  3.3 Example 37
  3.4 Conclusion 38

III CONTRIBUTIONS 39
4 OUR APPROACH: SOCIAL SCORING 41
4.1 Social scoring 43
4.1.1 Score propagation 43
4.1.2 Trust 45
4.1.3 Correlation 46
4.2 Confidence 47
4.2.1 Confidence coefficients 48
4.2.2 Confidence aggregation 52
4.2.3 Confidence propagation 53
4.3 Default Score 55
4.3.1 Computation of a default score 55
4.3.2 Required knowledge for a default score 57
4.3.3 Confidence on a default score 58
4.4 CoTCoDepth Social Scoring 59
4.4.1 Definition 59
4.4.2 Example 59
4.5 Conclusion 61
5 EVALUATION 63
5.1 Campaigns 65
5.1.1 Training set 65
5.1.2 Leave one out 66
5.2 Dataset 67
5.2.1 Epinions 67
5.2.2 Flixster 68
5.2.3 Appolicious 69
5.3 Implementation 70
5.3.1 CoTCoDepth Scorer 70
5.3.2 Evaluation 71
5.3.3 Views 71
5.3.4 Metrics 72
5.4 Influence of k and connectivity degree 72
5.4.1 Epinions: Alchemy dataset 72
5.4.2 Appolicious 75
5.4.3 Flixster 76
5.4.4 Conclusion 77
5.5 Comparison with existing approaches 78
5.5.1 Scorers characteristics 78
5.5.2 All actors 79
5.5.3 Cold start users 80
5.6 Conclusion 80
6 HEURISTICS 83
6.1 Heuristics evaluation protocol 85
6.2 Extended similarity 86
6.2.1 Definition 86
6.2.2 Evaluation 87

6.3 Relative scoring . . . 87
  6.3.1 Relative score propagation . . . 88
  6.3.2 Relative scoring evaluation . . . 88
6.4 A New Hop . . . 89
  6.4.1 Score propagation with hops . . . 90
  6.4.2 Hops evaluation . . . 90
6.5 Friends selection . . . 91
  6.5.1 Random friends selection . . . 92
  6.5.2 Random raters selection . . . 93
  6.5.3 Weight influence . . . 96
  6.5.4 Conclusion . . . 98
6.6 Expertise . . . 98
  6.6.1 Friends expertise . . . 98
  6.6.2 Global expertise . . . 99
  6.6.3 Expertise evaluation . . . 99
6.7 Conclusion . . . 100

IV CONCLUSION 101
7 CONCLUSION 103
7.1 Contributions . . . 103
7.2 Discussion . . . 106
7.3 Perspectives . . . 107

V APPENDIX 111
A MANAGING CYCLES 113
A.1 Cycles in score propagation . . . 114
  A.1.1 Ping-pong cycle . . . 114
  A.1.2 Loop . . . 114
  A.1.3 Duplicate scores . . . 115
A.2 Extended formula . . . 116
A.3 Evaluation of the extended formula . . . 116
B DATASETS 119
B.1 Rich Epinions Dataset . . . 120
  B.1.1 Epinions dataset extraction . . . 121
  B.1.2 Dataset structure . . . 122
  B.1.3 Statistics . . . 122
  B.1.4 Evaluation with this dataset . . . 126
B.2 Appolicious dataset . . . 126
  B.2.1 Appolicious dataset extraction . . . 127
  B.2.2 Dataset structure . . . 127
  B.2.3 Statistics . . . 128
B.3 Conclusion . . . 129
C PROTOTYPES 131
C.1 Scars prototype . . . 132
  C.1.1 Usage . . . 132
  C.1.2 Modules . . . 133
  C.1.3 Scorer . . . 134

- C.1.4 Evaluation . . . 136
- C.1.5 Conclusion . . . 137
- C.2 P2P prototype . . . 138
  - C.2.1 Implementation . . . 138
  - C.2.2 Results . . . 138
  - C.2.3 Conclusion . . . 139

BIBLIOGRAPHY 141

LIST OF FIGURES

Figure 1 Graph example 30
Figure 2 Simple graph example 31
Figure 3 Digraph example 32
Figure 4 Bipartite graph example 33
Figure 5 Weighted graph example 34
Figure 6 Other examples 34
Figure 7 Social, trust and similarity networks and ratings example 38
Figure 8 $k-$Depth Social Scoring Example 45
Figure 9 Trust Example 45
Figure 10 Correlation Example 46
Figure 11 Size confidence depending on number of friends' scores 49
Figure 12 Freshness confidence depending on scores age with different λ 51
Figure 13 Confidence Example 54
Figure 14 CoTCoDepth Example 60
Figure 15 Distribution on Alchemy 68
Figure 16 Distribution on Flixster 69
Figure 17 Distribution on Appolicious 70
Figure 18 Influence of training set size on coverage using Alchemy 73
Figure 19 Influence of connectivity degree on coverage using Alchemy 74
Figure 20 Influence of connectivity degree on precision using Alchemy 74
Figure 21 Influence of connectivity degree on coverage using Appolicious 75
Figure 22 Influence of connectivity degree on precision using Appolicious 76
Figure 23 Influence of connectivity degree on coverage using Flixster 77
Figure 24 Influence of connectivity degree on precision using Flixster 77
Figure 25 Influence of correlation on precision using Alchemy 87
Figure 26 Absolute vs. Relative scores on Alchemy . . . 89
Figure 27 $k-$Depth Social Scoring with Hops; Example with $\alpha = 0.5$ 90
Figure 28 Influence of α on the RMSE, using hops 91
Figure 29 Random friends selection evaluation using Alchemy 93
Figure 30 Random friends selection evaluation using Flixster 94

Figure 31 Random raters selection evaluation using Alchemy 95
Figure 32 Random raters selection evaluation using Flixster 96
Figure 33 Weight influence compared to random raters selection 97
Figure 34 Cycle with 2 actors 114
Figure 35 Cycle with 3 actors 114
Figure 36 Cycle with 4 actors 116
Figure 37 Database schema of the dataset 123
Figure 38 Trust distribution 124
Figure 39 Ratings count distribution 124
Figure 40 Database schema of the anonymised dataset . 128
Figure 41 Following distribution 129
Figure 42 Ratings count distribution 129

LIST OF TABLES

Table 1 State of the art on recommender systems . . . 27
Table 2 Results for all actors on Epinions 79
Table 3 Results for cold start users on Epinions 80
Table 4 RMSE with relative or absolute score depending on k 89
Table 5 Results for expertise heuristics on Epinions . . 100
Table 6 Summary of recommender systems characteristics 107
Table 7 Parenting strategies evaluation 117
Table 8 Statistics depending on user characteristics . . 123
Table 9 Views distribution 125

Part I

INTRODUCTION

1

INTRODUCTION

> *Computers are useless. They can only give you answers.*
> *— Pablo Picasso*

1.1 CONTEXT

The Internet gives users easy and immediate access to a lot of resources: web pages, video on demand, music streaming, services, etc. Among this abundance of items, information overload is an ever growing problem.

Recommender systems are one solution classically proposed to cope with this problem [SKR99]. They offer a top-k list of items to the users and reduce the number of possibilities in order to ease the user choice process. Recommender systems are often classified in two types: content-based and collaborative filtering recommender systems.

Content-based recommender systems compare items characteristics in order to find similar items to the ones liked by users. They build users' preferences based on the items they liked or bought, as on the Pandora [1] music website.

Collaborative filtering recommender systems rely on relations between users to predict items that could fit users' interests based on related users ratings. Those relations are often modeled as user-user matrices or user-user graphs. They are usually similarity relations [AT05]. Collaborative filtering is notably used on Amazon [2].

Trust-based recommender systems are collaborative filtering systems built on user relations that express the trust users have on the opinion of others [OS05, MA07a, MKL09]. A typical example is the Epinions [3] website [RD02], where they recommend items liked by trusted users. A trust link means that a user believes on the usefulness of the recommendation of a trusted user. The main problems of these systems rely on the ratings sparsity and on the so-called cold start users who have no or very few connections in the graph. To remedy sparsity, trust-based recommender systems increase the number of users relations by propagating trust relations based on some transitivity property.

Social-based recommender systems take into account social relations and social information in their predictions. In social network, relations are explicitly decided by end-users (even when recommended by the system like in Facebook [4]). In such a context , we believe relations cannot be implicitly propagated by the system. Recommender systems should find other solutions to cope with sparsity.

---

[1] `http://www.pandora.com`
[2] `http://www.amazon.com`
[3] `http://www.epinions.com`

Web 2.0 and social networks provide a lot of new content on the Internet. Recommender systems take into account those new pieces of information in their algorithms in order to provide more personalized recommendations. They process users' information such as reviews, Facebook "Likes", Google+ "+1", tweets, etc. in order to infer users preferences on items.

In this win-win scenario, users willingly share their profiles in order to get relevant and accurate recommendations. Logging into a website using a Facebook account allows the website to use the user's Facebook profile in order to infer user's preferences. For example, in Flixster, users can add their Facebook friends automatically; they also rate movies and build their movies collections. In return they receive movies recommendations.

1.2 MOTIVATIONS

Users usually know what they share or not with a system. Most pieces of information are explicit: their profile, their Facebook friends, etc. However, it is much harder to know what the system infers from that knowledge. Some studies tend to show that inferences can be of unexpected accuracy for private information.

For example, in psychology, the "Big Five" factors of personality are dimensions of personality that are used to describe human personality. These factors are openness, conscientiousness, extraversion, agreeableness and neuroticism [Gol90]. Although these factors should be private and confidential, recent studies have shown that it is possible to infer them from Facebook [BKG+12] or Twitter [QKSC11] data. More important, [QKSC11] predicts personality with a good accuracy using only public data available on Twitter.

To avoid this situation and for privacy concerns, the trend is to limit the transmission of information on a user only to authorized related users. Efforts are conducted to decentralize social networks infrastructures (*e.g.* Diaspora [5]), so as to impede their owners to gather all information on people. The FreedomBox [6] concept imagines that each user will have his/her own node hosted in his/her own web server. These new architectures intend to decentralize knowledge.

But many recommender systems rely on global knowledge to predict missing ratings: they use the whole user graph. For instance, collaborative filtering systems predict ratings *a priori* using a costly computational process that builds the whole users' similarity graph from all users' profiles. Such a mechanism requires all ratings to be fully known by the system and therefore cannot be used in architectures that decentralize data. Moreover users have to share personal data with the system in order to get recommendation.

---

[4] `http://www.facebook.com`
[5] `http://joindiaspora.com/`
[6] `http://wiki.debian.org/FreedomBox`

On the other side, local approaches use knowledge limited to a subpart of the users' graph and profiles. Each rating calculation is processed using the requesting user's profile and the set of directly linked users, the system does not require to know personal data from all users. This constraint enhances privacy, requires lighter processing and complexity is independent of the number of users. They are also feasible on decentralized architectures. Local approaches do not require *a priori* processing of predictions; they can predict ratings on demand. Local approaches are *in fine* compatible with P2P architectures and allow privacy strategies.

That is why we propose in this thesis a social-based recommender system that relies on local knowledge in order to:

- enhance privacy by limiting knowledge disclosure,
- be implementable in P2P architectures with decentralized data on low-resource peers,
- keep good results in terms of coverage and accuracy, especially for cold start users.

1.3 CONTRIBUTIONS

Our system relies on the users social network without modifying it. Users can explicitly define whom they want to communicate with. It shares information only between authorized related users, therefore it could be implemented as an additional service on top of existing social networks such as Facebook, Diaspora or LastFM.

Besides, our local approach provides a P2P compatible architecture. It respects the following decentralisation definition:

- *Decentralized computation*: computations are partially performed on several nodes. We assume these nodes not to be powerful machines.
- *Decentralized data storage*: data and preferences are distributed throughout the network, typically on peers computers.

In chapter 4, we propose a recommender system that propagates ratings locally in a users' network without propagating relations:

- In order to predict ratings, users asks their friends their scores on items. Friends propagate their scores in the network step by step, until reaching the original requester with a maximum depth propagation (section 4.1). Scores are either a rating if a user has one, or an aggregation of friends' scores (section 4.1.1). We weight relations by explicit trust (section 4.1.2) or correlation (section 4.1.3).

- Our system associates confidence with each prediction during the propagation (section 4.2). This coefficient integrates: links weights, number of aggregated scores, variance of the aggregated scores, users distance and score freshness. The higher the confidence, the more accurate the prediction should be.
- We propose two default scoring strategies for sparse networks adapted to our approach. They are based on classical default rating approaches (section 4.3).
- We finally describe our recommender system, CoTCoDepth, in section 4.4 that integrates all of the above.

We have used several datasets to evaluate our approach and compare it with five collaborative filtering and trust-based recommender systems in chapter 5:

- Section 5.1 defines the campaigns used in our evaluation.
- Section 5.2 describes the three evaluation datasets: Epinions Alchemy, Flixster and Appolicious.
- Section 5.3 explains some implementation details of the evaluation, in particular the three metrics used: coverage, precision / RMSE and f-measure.
- Section 5.4 analyses our approach regarding the three datasets. It focuses on precision and coverage, with or without default scores.
- Section 5.5 compares our approach with the state of the art in trust-based recommender systems and with classical collaborative filtering algorithms.

We show that a well chosen limited vision of the ratings remains efficient in ratings prediction, with sparse or dense datasets. Our approach shows good results even when users are weakly connected and/or when they did not rate many items, whereas they are usually the hardest users to recommend.

Furthermore, we have defined several heuristics pluggable in our system in chapter 6:

- Since friends and similar users are usually disjoint, we propose an extended version of the correlation coefficient that computes much more coefficients between friends.
- Relative scoring removes actors ratings bias before propagating scores. Doing so, less information is shared in the network and rating behaviour is taken into account (section 6.3).

- *Hopping propagation* propagates more scores in the network, enhancing the accuracy and lowering ratings disclosure. However, it also floods the network with more requests (section6.4).
- Section 6.5 adapts the score propagation to P2P architectures by limiting the number of requests in the network.
- Finally, section 6.6 adds expertise in the friends selection.

Before describing our approach, we first propose in chapter 2 a state of the art in recommender systems. Section 2.1 describes content-based recommender systems whereas section 2.2 describes collaborative filtering recommender systems. Section 2.3 focuses on trust-based recommender systems since our approach is quite similar to them. Finally, section 2.4 is about social-based recommender systems.

We then specify some definitions and vocabulary in chapter 3. Section 3.1 gives graph theory definitions. Section 3.2 employs the latter to define the vocabulary used to describe our approach. Finally, section 3.3 describes our main example that illustrates our approach defined in chapter 4, evaluated in chapter 5 and altered by heuristics in chapter 6. After that we conclude and discuss our work in chapter 7.

We ultimately propose the following appendices:

- In appendix A, we explain how we manage cycles in our approach.
- In appendix B, we describe the two datasets that we have built during this thesis.
- In appendix C, we portray the two prototypes we have implemented to evaluate our approach.

Part II

PREAMBLE

2

STATE OF THE ART ON RECOMMENDER SYSTEMS

> *Every man wishes to be wise, and they who cannot be wise are almost always cunning.*
> *— Samuel Johnson*

Contents

2.1 Content-based recommender systems 12
  2.1.1 Description of textual content 13
  2.1.2 Description of other kind of content 14
  2.1.3 User profile 14
  2.1.4 Conclusion 15
2.2 Collaborative filtering recommender systems . . . 16
  2.2.1 Centralized version 17
  2.2.2 Decentralized version 17
  2.2.3 Conclusion 18
2.3 Trust-based recommender systems 18
  2.3.1 TidalTrust 20
  2.3.2 MoleTrust 21
  2.3.3 TrustWalker 21
  2.3.4 Conclusion 23
2.4 Social-based recommender systems 23
  2.4.1 SocialMF 24
  2.4.2 Hoens et al. [HBC10] 25
  2.4.3 Conclusion 25
2.5 Conclusion 26

The following state of the art details the two main recommender systems categories: content-based and collaborative filtering recommender systems as well as two specific kinds of collaborative filtering namely trust-based and social-based recommender systems. Those categories dispose of several kinds of data on users and / or items.

This chapter analyzes how those systems take into account privacy and ratings disclosure. It also focuses on how those approaches can be compatible with decentralized architectures, and particularly peer-to-peer ones. Finally, we target their performance with cold start users, since they usually are the hardest to recommend items to.

Section 2.1 explains how content-based approaches predict preferred items based on the inner features of the items.

Then section 2.2 depicts collaborative filtering recommender systems, which predict ratings based on the ratings made by other users, especially similar users. Firstly, centralized versions are detailed in section 2.2.1. They access a global knowledge and compute similarity between all users. Secondly, section 2.2.2 introduces decentralized collaborative filtering recommender systems. Here, data are decentralized, meaning that no node knows all the ratings. Therefore the similarity computation is not straightforward.

Section 2.3 focuses on trust-based recommender systems. Those recommender systems specialize collaborative filtering ones, using trust instead of similarity between users. In order to compare our trust-based compatible approach, we detail three of the main trust-based approaches: TidalTrust (section 2.3.1), MoleTrust (section 2.3.2) and TrustWalker (section 2.3.3).

Finally, section 2.4 describes social-based recommender systems. Similar to trust-based recommender systems, they rely on social relations to process recommendation. Section 2.4.1 introduces SocialMF, a matrix factorisation technique that incorporates trust propagation. Section 2.4.2 portrays Hoens et al. approach based on propagation of encrypted ratings in a social network.

2.1 CONTENT-BASED RECOMMENDER SYSTEMS

Content-based recommender systems recommend items that are similar to the ones that the user has liked in the past [RRSK11]. For example in order to compute similarity between movies or musics, multimedia content-based recommender systems compare items features, such as movie or music genre, producer, artist, subject matter, date, etc. Items can be products, multimedia content, documents, people, company, etc.

Content-based approaches emerged from information retrieval and information filtering researches [AT05, BC92]. Therefore, many systems focus on recommending items containing text, such as documents. The use of metadata also enhances integration of content-based recommender systems with any kind of items.

Content-based approaches rely on user *profiles* containing information about users' tastes, preferences and needs. Those profiles are explicitly defined by users and/or implicitly learned from their behaviour.

Items profiles contain information on items, representing their contents. They can be defined explicitly by users or experts, or extracted from items features.

We detail them in the following subsections. First, section 2.1.1 focuses on textual documents and how to extract profiles from them. Then, section 2.1.2 enlarges it to build profiles for different kinds of items. Finally, section 2.1.3 lists some approaches to build user profile from items profiles, and then to recommend items to a user, based on those profiles.

2.1.1 *Description of textual content*

Text-based recommender systems find the most important words in a document and use them to construct the document profile. [BS97] propose a system that represents textual document with the 100 most important words.

The importance of words in a document can be determined by several metrics [BYRN99]. Different models have been defined in the literature. One can cite the boolean model, the probabilistic model and the vectorial model [SM86]. We concentrate here on the vectorial model. Commonly used in information retrieval works, the *term frequency/inverse document frequency* (TF-IDF) defines a word importance based on its occurrence frequency in documents:

Definition 1: $\mathsf{TF}_{i,j}$**.** *Let* w_i *be a word,* d_j *a document,* $f_{i,j}$ *the occurrence frequency of* w_i *in* d_j*. The normalized term frequency of word* w_i *in document* d_j *is defined as:*

$$\mathsf{TF}_{i,j} = \frac{f_{i,j}}{\max_k f_{k,j}}$$

Where the maximum is computed regarding the frequencies $f_{k,j}$ of all words w_k in d_j.

Definition 2: IDF_i**.** *Let* N *be the total number of documents and* n_i *the number of documents containing* w_i*. The inverse document frequency of word* w_i *is defined as:*

$$\mathrm{IDF}_i = \log \frac{N}{n_i}$$

The TF-IDF weight of word w_i in document d_j is computed as:

$$\omega_{i,j} = \mathsf{TF}_{i,j} \times \mathrm{IDF}_i$$

The objective of TF/IDF is to identify important words, *i.e.* word that are frequent (TF) but discriminating (IDF).

We then define documents' profiles as vectors of words weights:

Definition 3: document profile. *Let* $\omega_{i,j}$ *the weight of word* w_i *in document* d_j*. The profile*(d_j) *of the document* d_j *is defined as:*

$$\textit{profile}(d_j) = (\omega_{1,j}, \cdots, \omega_{k,j})$$

First approaches of content-based recommender systems focused on text, they serve as a basis for the description of other kind of content.

2.1.2 *Description of other kind of content*

Regarding other items than textual documents, some works focus on metadata associated with the items [WL02]. These metadata can be manually defined or automatically extracted from the items.

[PFW05] work on similarity computed between musics, based on audio spectral measures and fluctuation patterns. They show that this improves recommendation accuracy, however it is music specific and cannot be applied on other fields.

In [DMO+12], Di Noia et al. propose a semantic approach based on linked data included in the datasets from the Linked Open Data cloud [BHBL09]. Items profiles are build using semantic descriptions of items.

We see that an item can be described by its metadata, features or semantic descriptions. Either ways, that description is used to build the item's profile.

Items' profiles are still represented by vectors, where the dimension number is the number of metadata, features or concepts. For each dimension, a weight is associated regarding the item description.

The profile *profile*(i) of the item i consists then of a vector of metadata / feature / concept weights, as in definition 3.

2.1.3 *User profile*

Let *profile*(i) be the profile of item i. It contains usually features extracted from the item, represented as a vector. This profile is used as a representation of the item in order to find items suitable for a target user.

Let *userProfile*(u) be the profile of user u. It contains tastes and preferences of the user. The user profile is built using items profiles rated by the user. It is usually a vector of weighted features: the higher the weight, the more the user likes the feature. The size of the vector correspond to the number of features.

Several approaches exist in order to build the user profile from items profiles. In [BS97], the authors compute the user profile as an average vector from items profiles evaluated by the user.

To make recommendations to users, distance is computed between users' profiles and items' profiles in order to find items that are more likely to be preferred by users. The distance between user u and item i, represented by their profiles $\vec{p_u}$ and $\vec{p_i}$, is usually computed thanks to the cosine similarity measure [BYRN99]:

$$distance(u, i) = 1 - \cos(\vec{p_u}, \vec{p_i}) \tag{1}$$

With:

$$\cos(\vec{u}, \vec{v}) = \frac{\vec{u}.\vec{v}}{\|\vec{u}\|_2 \times \|\vec{v}\|_2} = \frac{\sum_{k=1}^{n} u_k \times v_k}{\sqrt{\sum_{k=1}^{n} u_k^2}\sqrt{\sum_{k=1}^{n} v_k^2}} \tag{2}$$

Then, items with profiles close to a user profile will be recommended to the user.

Items similarities are also computed using items profiles. In order to compute similarity between items i and j, represented by their profiles $\vec{p_i}$ and $\vec{p_j}$, the cosine similarity measure is also used:

$$sim(i, j) = \cos(\vec{p_i}, \vec{p_j}) \tag{3}$$

Then, items similar to an item liked by a user will be recommended to the user.

2.1.4 *Conclusion*

Content-based recommender systems offer a way to recommend items based on their features. Some approaches start to take into account social tags in their models [CDHP11]. However they suffer some drawbacks as described below.

Although it works well with text documents, extracting features from items is not straightforward. Content-based approaches are more likely to be deployed with specific items, such as movies or musics. Moreover some kinds of items, such as poems or jokes, do not contain enough words to extract discriminant profiles [PB07].

In addition, content-based approaches recommend items similar to users tastes or preferences. Therefore they lack serendipity and recommend the same kind of items [HKTR04].

Moreover, cold start users, who have not rated enough items to construct a reliable profile, receive in general no accurate recommendation or no recommendation at all.

Pilászy et al. show in [PT09] that even a few ratings are more valuable than metadata. Content-based and collaborative filtering recommender systems work well together and hybrid systems perform usually better.

Regarding privacy, items profiles are build without users data, therefore it discloses no information about users' profiles. In order to build a user profile, content-based recommender systems access items liked by the user and aggregate their profiles. This could be done locally on the user's computer.

2.2 COLLABORATIVE FILTERING RECOMMENDER SYSTEMS

Collaborative filtering systems try to predict items ratings for a particular user based on the items previously rated by other users [AT05]. They rely on two pieces of information: users' profiles and a user graph.

Users' profiles contain ratings from users to items, purchased history, interest, etc. The user graph contains users as nodes and links between them, *i. e.* relations between users. Typically, those links are correlation (*aka.* similarity) coefficients computed using users ratings [BHK98].

To recommend items to a specific user, the system uses the profiles of similar users, *i. e.* users directly linked to the user in the user graph.

For example in a movie recommendation application, in order to recommend movies to a target user, a collaborative filtering system tries to find users that have similar tastes in movies. Then, the movies liked by those users are recommended to the target user.

To predict a rating for a specific user, collaborative filtering recommender systems aggregate other users' ratings with the following functions [AT05]:

$$r_{a,i} = \frac{\sum_{a' \in A_i} \omega_{a,a'} \times r_{a',i}}{\sum_{a' \in A_i} \omega_{a,a'}} \quad (4)$$

$$r_{a,i} = \overline{r_a} + \frac{\sum_{a' \in A_i} \omega_{a,a'} \times (r_{a',i} - \overline{r_{a'}})}{\sum_{a' \in A_i} \omega_{a,a'}} \quad (5)$$

Where $r_{a,i}$ is the rating given by user a to item i, A_i is the set of users having rated item i and $\omega_{a,a'}$ is a weight between users a and a', typically a similarity coefficient.

In equation 5, $\overline{r_a}$ is the ratings mean of user a, it is considered as a bias: some people have a high ratings mean, while others have a low ratings mean. Which means that regarding a range of ratings in $[\![1, 5]\!]$, some users define 3 as a good evaluation whereas some others define 3 as a bad evaluation. This equation tries to take into account users' rating subjectivity in predictions.

2.2.1 *Centralized version*

In this document, we call UserBasedCF (respectively ItemBasedCF) the collaborative filtering algorithm defined in eq.4 using the Pearson's correlation coefficient ρ between two users' ratings (resp. two items' ratings) as ω similarity coefficient [BHK98]. Lets consider r and p two lists of N ratings, r_n (resp. p_n) is the n^{th} rating of the list r (resp. p), $\bar{r}$ (resp. $\bar{p}$) is the mean of the ratings contained in r (resp. p). The Pearson's correlation coefficient is computed as follow:

$$\rho = \frac{\sum_{n=1}^{N}(r_n - \bar{r})(p_n - \bar{p})}{\sqrt{\sum_{n=1}^{N}(r_n - \bar{r})^2} \times \sqrt{\sum_{n=1}^{N}(p_n - \bar{p})^2}} \tag{6}$$

UserBasedCF (resp. ItemBasedCF) covers all users (resp. items) and relies on global knowledge on their profiles in order to compute similarities between them. It constructs a similarity graph and selects similar users (resp. items) to predict missing ratings. Both approaches require ideal conditions where all ratings are known by the systems and all users and items can be easily identifiable.

When two users have not enough ratings in common to compute similarity, some default ratings may be introduced [BHK98]. Those can be a neutral rating, the item mean rating or the user mean rating. Default ratings allow the system to cope with sparsity.

Classical collaborative filtering recommender systems usually require centralized architectures in order to compute similarity between items or users. However decentralized approaches have emerged in order to provide recommendation in P2P architectures.

2.2.2 *Decentralized version*

In this section, we focus on decentralised user-based collaborative filtering recommender systems, *i.e.* systems with decentralised data storage and decentralized computation, as defined in section 1.3.

The first step in collaborative filtering is to assign neighborhood to users. Reaching all users in order to compute neighborhood is not efficiently achievable in a decentralized system.

In epidemic protocols (*aka. gossip protocols*), a peer accesses a limited view of the network. This view initially contains random users and their profiles, selected by a Random Peer Sampling service (RPS) [JVG$^+$07]. Similarity is computed between the peer and each user in the view. Only the n more similar users are maintained in the view.

Using a clustering protocol, the view is updated continually in order to get the most similar users in the view. A peer periodically selects a gossip target from its view and exchanges view information with the target. The peer compares the new candidates with the existing ones and keeps only the n most similar users as a new version of the view [JB06, KLMT10]. The RPS is still used in order to randomly add new users in the view and lower the risk of network partitioning.

Finally, the view converges until finding locally an optimal neighborhood, containing similar users. Afterward, recommendation follows collaborative filtering approaches, asking similar users their ratings and computing a weighted mean.

Neighborhood views strongly depend on similarity metrics. Different metrics will return different views. Therefore one single metric could hardly optimize all users views: metrics selection should depend on the users' profiles. In [KT12], the authors show that selecting the appropriate metric for each user increases recommendation recall from 78 % to 85.2 %.

Decentralized user-based collaborative filtering recommender systems do not suffer from scalability issues as their centralized counterpart, since each peer computes locally recommendations for its associated user. However similarity is locally optimized, missing some similar users in the neighborhood computation. And peers communication can induce some overhead that should be evaluated.

2.2.3 *Conclusion*

Although collaborative filtering recommender systems are successfully used by most e-commerce websites.

They are not well-protected against malicious [MHN07] or peculiar [SFHS07] users and hardly cope with cold start users (who rated few or no items) or with the overall sparsity of existing ratings [LB09]. Finding similar users requires heavy computation, for both centralized and decentralized architectures. Finally, users share their profiles with either a server (in centralized systems) or unknown peers (in decentralized systems), thus limiting privacy.

In order to cope with all those drawbacks, some systems focus on trust instead of similarity to build user graphs.

2.3 TRUST-BASED RECOMMENDER SYSTEMS

Trust is a polysemic term for which [Wil93, Has10] propose several definitions. One of the earlier notable definitions of trust is formulated by the social psychologist Morton Deutsch [Deu62]. The definition states that when:

1. "the individual is confronted with an ambiguous path, a path that can lead to an event perceived to be beneficial ($Va+$) or to an event perceived to be harmful ($Va-$);

2. he perceives that the occurrence of $Va+$ or $Va-$ is contingent on the behavior of another person; and
3. he perceives the strength of $Va-$ to be greater than the strength of $Va+$.

If he chooses to take an ambiguous path with such properties, I shall say he makes a trusting choice; if he chooses not to take the path, he makes a distrustful choice."

When focusing on recommender systems, trust is generally interpreted as a belief of the usefulness of someone's recommendation [VCC11]. More precisely, global and local trust metrics exist.

For example, PageRank [PBMW99] is a global metric asserting the usefulness of a document based on links going in or out this document. In eBay, sellers reputation is an aggregation of votes providing a global rate saying if a seller is trustworthy or not.

Epinions website is however providing users tools to build their own web-of-trust, *i.e.* a set of trusted or distrusted users. In this situation, local trust is defined from one user to another user.

In this thesis, trust is defined as the local belief of one user in the usefulness of information provided by another user [LB09]. Trust can theoretically be positive, null or negative. In real life systems, trust with negative values are rarely used.

Trust-based recommender systems invite users to state that they trust the ratings expressed by other users [OS05, MA07a, MKL09].

They address collaborative filtering recommender systems' drawbacks by using trust instead of similarity. Cold start users do not need to rate items to start using the system, they need to trust other users [MA07a, PK09]. Security against malicious users is improved since trust relations imply that users know their direct relations.

Traditional trust-based recommender systems initially rely on trust values to provide the ω weight defined in eq.4 and 5. The ratings prediction equations become then [VCC11]:

$$r_{a,i} = \frac{\sum_{a' \in T_a} t_{a,a'} \times r_{a',i}}{\sum_{a' \in T_a} t_{a,a'}} \tag{7}$$

$$r_{a,i} = \overline{r_a} + \frac{\sum_{a' \in T_a} t_{a,a'} \times (r_{a',i} - \overline{r_{a'}})}{\sum_{a' \in T_a} t_{a,a'}} \tag{8}$$

With T_a the set of trusted user of a who evaluated i and for which the trust value $t_{a,a'}$ exceeds a given threshold.

However, trust-based recommender systems have to deal with the sparsity of trust networks: in Epinions [RD02], a user trusts in average 10 users. To do so, they usually propagate trust in the network [Gol05]. Trust is thus considered as a transitive property, so that the graph can be automatically updated by propagating inferred trust values from previous calculations [HWS09, MA07a]: if a trusts b and b trusts c, a new trust value from a to c is defined and a new link between users a and c is added in the graph. Several iterations are performed in order to explore the graph up to a given depth.

2.3.1 *TidalTrust*

Golbeck et al. propose a recommender algorithm based on equation 7 in [Gol06]. The main novelty of this algorithm lies in the way trust values are inferred with the trust metric TidalTrust.

In their paper, the authors define the following guidelines derived from trust networks analysis:

1. for a fixed trust value, shorter paths have a lower error,
2. for a fixed path length, higher trust values have a lower error.

In order to satisfy the first observation, without sacrificing coverage by using too short paths, they define a variable path length depending on users: the shortest path length that is needed to connect the target user u with a user v that has rated the item (*aka.* rater) becomes the path depth of the algorithm for the user u.

To address the second observation, they define a trust threshold that filters users participating in the prediction. They incorporate a value that represents the path strength, *i. e.* the minimum trust value on the path. They then compute the maximum path strength over all paths leading to the raters. This maximum becomes the threshold.

The full algorithm for inferring trust of user u on user v is therefore:

$$t_{u,v} = \frac{\sum\limits_{u' \in T_u^+} t_{u,u'} \times t_{u',v}}{\sum\limits_{u' \in T_u^+} t_{u,u'}} \tag{9}$$

With T_u^+ the set of trusted users for whom u's trust value exceeds the given threshold. TidalTrust is recursively computed for all users that are the first link on the shortest path from u to v. This value is not symmetric $t_{u,v} \neq t_{v,u}$

Golbeck et al. provide a modified breadth-first algorithm to compute trust between a user u and a set of raters. As explained in [GH06], the system first searches for raters that u knows directly. If no direct connection exists from the user to any rater, the system moves one step out to find connections from the user u to raters of path length 2. This process is repeated until a path is found.

The ratings of all raters at that depth are considered, using equation 7, the trust values between them and the user u are computed using equation 9.

TidalTrust takes into account shortest paths with a modified breadth first search in the trust network. Since it only uses information from raters at the nearest distance, it may loose a lot of valuable ratings from users a little further apart in the network.

It also requires an extended-local knowledge of the trust network, since all paths surrounding users must be followed to compute the final trust values. Moreover, the whole network is explored until raters have been found, without limiting the propagation. Finally, it requires a global orchestration when selecting the raters depth.

2.3.2 *MoleTrust*

Massa et al. propose a recommender algorithm based on equation 8 in [MA07a]. They incorporate a new trust metric, called MoleTrust. Before computing trust, cycles in the trust network are removed, then each user only needs to be visited once to obtain a trust prediction.

After removing cycles, the trust network becomes a directed acyclic graph. Therefore the trust prediction $t_{u,v}$ can be obtained by performing a simple graph walk: first trust values of users at distance 1 are computed, then trust values of users at distance 2, etc.

In order to compute a trust value between users u and v, MoleTrust proceeds in a similar way as TidalTrust in equation 9, with a different selection of users in T_u^+: MoleTrust considers all users connected to u. In order to stop the trust propagation at some point, they define a trust horizon, *i.e.* a maximum depth propagation k, being the maximum distance between u and v [MA07b]. Beyond that distance, trust is not computed.

MoleTrust predicts the trust value of a source user to a target user by gradually propagating trust in the user graph, up to a given depth k. If more than one trust path links two users, the mean of all computable trusts is used. This approach thus requires an extended-local knowledge of the trust network, since all the paths surrounding those users must be followed to compute the final trust value. Moreover, the whole user graph is explored up to depth k in order to predict ratings.

The authors use TrustAll as a baseline of their algorithm. TrustAll is an algorithm taking into account all users ratings, as if the user trusted every one else: it simply computes items' ratings mean as predictions.

2.3.3 *TrustWalker*

Jamali et al. propose TrustWalker, a random walk model combining trust-based and item-based recommendation in [JE09]. Unlike the previous trust-based approaches, TrustWalker performs random walks in the trust network.

However they consider not only ratings of the target item, but also those of similar items. The probability of using the rating of a similar item instead of a rating for the target item increases with the length of the walk.

The random walk performs the search in the trust network. To recommend a rating for a source user u_0 on target item i, they perform random walks on the trust network, each starting at u_0 to find a user having expressed rating for i or items similar to i. A random walk consists of going from a trusted user to another trusted user, until a rating is found. At each step, if no rating is found, only one trusted user is randomly selected as the next step of the walk. Each random walk returns a rating. The aggregation of ratings returned by different random walks are considered as the predicted rating.

The probabilistic item selection is based on items similarities computed using a Pearson correlation coefficient on items' ratings, as in [SKKR01]. They consider only positive similarity values. At each step, if the current user does not have any rating on the target item i, the probability to stop the walk and return the rating of a similar item j is proportional to the similarity of i and j.

Therefore, at each step of the random walk, if the current user has a rating on the target item, it is returned. Otherwise there is a probability to select a similar item's rating, *c.f.* below, or to select a trusted user in order to go further into the walk. If no item is selected at a maximum depth k, the walk stops and a similar item's rating is returned. In their paper, $k = 6$.

They perform several random walks to be able to get a more reliable prediction. In order to consider the prediction stable enough to stop the random walks, they compute the variance in the results of all the walks as follows:

$$\sigma_T^2 = \frac{\sum_{i=1}^{T} (r_i - \bar{r})^2}{T} \tag{10}$$

With r_i the predicted rating of the i^{th} random walk and $\bar{r}$ the average of the ratings returned by the T first random walks. T is the number of random walks performed until now to compute the prediction. Since the values of ratings are in the finite range $[1,5]$, they prove that σ_T^2 converges to a constant value. So they terminate TrustWalker if $|\sigma_{i+1}^2 - \sigma_i^2| \leqslant \epsilon$. In they paper, they define $\epsilon = 0.0001$.

They also define a constant threshold of 10 000 for the maximum number of unsuccessful random walks, *i.e.* the stop condition described above is not reached, after that they consider the rating as uncovered.

Finally, once the prediction is computed, they associate a confidence value on the prediction. Most existing recommender systems do not provide users a confidence on their predictions.

They use σ_T^2 defined above in order to compute this confidence:

$$confidence = 1 - \frac{\sigma_T^2}{max_{\sigma^2}} \tag{11}$$

Where max_{σ^2} is the maximum possible variance for the results. With ratings being in a finite range *Range*: $max_{\sigma^2} = \frac{Range^2}{4}$.

The use of item-based collaborative filtering during random walks significantly improves coverage regarding classical trust-based approaches. In addition, since only one trusted user is selected at each step of the walk, the complexity is not exponential with respect to the maximum depth of the walks.

However, this requires a global knowledge on items ratings since similarity computation is based on all items ratings. Moreover, the stop condition is really strict with $\epsilon = 0.0001$ and a maximum threshold of 10 000. It requires a lot of random walks in order to provide an accurate prediction.

The purely trust-based version of their approach is called RandomWalk. In this version, no item similarity is used, which means that only ratings of the target item are returned by random walks. It relies only on local knowledge. This is actually the only purely local trust-based approach we found in the literature.

2.3.4 *Conclusion*

Existing approaches offer recommendation to the cost of new relations between users or global knowledge on users profiles. Trust propagation augments the trust network *a priori* by predicting new relations from existing ones. It is not purely local, as it knows every path from a user to another. Therefore privacy cannot be ensured to the users of the system.

Moreover, cold start users are also the ones providing few trust relations, limiting trust propagation. TrustWalker proposes an innovative way to deal with those users by returning similar items' ratings as default ratings. But this only works if similarity is computable and if the users have rated similar items.

2.4 SOCIAL-BASED RECOMMENDER SYSTEMS

Social recommender systems rely on commonly used social networks such as Facebook, Twitter or Linkedin [JE10]. They do not have to cope with the sparsity problem because of the numerous existing relations of these networks instead of trust ones (to this date: 1 billion active users on Facebook with 140 friends in average [1]).

Some social-based recommender systems rely on the content of the information produced by the users during their interactions with their social networks (such as tweets, comments, likes, etc.) [MG11, CDHP11], whereas some other directly rely on the network structure (*e. g.* the graph of social relations) [JE10, HBC10, MZL+11, HC10].

We focus on the latter category since they allow the hybridization of trust and social approaches as we intend to. Social relations are different from trust relations. When a user u likes a review issued by another user v, u may add v to his/her list of trusted users. Trust generation is unilateral and does not require validation from the other side of the relation. This also indicates that user u does not need to even know user v in the real life. Social relations refer to mutual relations that surround us such as friends, classmates, colleagues, relatives, etc.

We first present a matrix factorization approach in section 2.4.1. Social-based matrix factorization approaches enhance the factorization with social relations [JE10, MZL+11].

We then describe a rating propagation approach in section 2.4.2. Such approaches aggregate immediate friends ratings and friends of friends ratings in the social network, similarly to trust based aggregation [HBC10, HC10].

2.4.1 *SocialMF*

Jamali et al. explore a model-based approach for recommendation in social networks in [JE10]. They employ matrix factorisation with trust propagation techniques on social relations.

Matrix factorisation techniques require global knowledge on the ratings matrix and heavy computations. Based on a global training set of ratings, this approach builds a predicting model. This model is then enhanced by social relations between users.

This approach is particularly effective with centralized architectures. Their evaluation shows that social information improves accuracy.

However it is hardly compatible with decentralized architectures. Moreover, since they use links propagation in the social networks, they add implicitly new links between users, without the control of the latter.

---

[1] Facebook statistics can be found at `http://newsroom.fb.com`

2.4.2 *Hoens et al. [HBC10]*

Hoens et al. provide a recommender system based on social network that does not modify it in [HBC10]. Ratings are propagated and aggregated through the social network up to a maximum depth, in a similar way as what we do.

Their approach takes into account privacy by encrypting ratings during the propagation. They use a semantically secure public-key homomorphic encryption scheme, which allows some operations on the underlying ratings. In particular, they use additively homomorphic encryption that has the following properties: given ratings r_1, r_2 and encryption algorithm Enc, $\mathsf{Enc}(r_1) \times \mathsf{Enc}(r_2) = \mathsf{Enc}(r_1 + r_2)$, which also implies that $\mathsf{Enc}(r_1)^c = \mathsf{Enc}(c \times r_1)$ for a positive integer c.

Homomorphic encryption requires heavy computation resources to aggregate ratings, on users sides. Moreover, the whole user tree from the requester to the children up to the defined depth is used in the computation. Therefore this approach is not deployable yet on mobile or low resource devices.

Users propagate ratings in the network. Ratings' weights are defined by users, which requires a lot of manipulation and is seldom made by average users. The cryptography prevent the system to compute similarity between users.

In addition, no confidence on the results is provided to users.

Finally, their work is only adapted to social networks: the heavy computation requirement limits the propagation to low depth, which provides high coverage only with dense social networks, like Facebook. With sparse networks, such as the trust network Epinions, this usually does not cover enough ratings.

2.4.3 *Conclusion*

Existing social-based recommender systems integrate social information in their algorithms. Social networks focus on social interactions: relations denote "friendship". It does not mean that friends' opinions are necessarily valuable for recommendation. However, evaluations show that this information improves accuracy and coverage.

SocialMF [JE10] use links propagation techniques, similar to trust propagation, to add new links in the network. It considers social networks as a kind of trust network. Resulting on the same drawbacks as with trust-based systems regarding ratings disclosure and privacy.

Hoens et al. [HBC10] propagate ratings in the social network. They propagate ratings through common friends instead of adding new links between users and friends of friends.

Unlike classical trust-based systems, we believe social-based recommender systems should not propagate social links since those relations are usually explicitly defined by users and have to be approved by them. Few are the works that consider social networks as not modifiable and that take into account users privacy.

2.5 CONCLUSION

This chapter presents a state of the art on recommender systems.

Content-based systems extract items characteristics in order to compare and recommend them. Although we have not proposed a content-based approach, they are compatible with our constraints and with our system.

Collaborative filtering systems provide recommendation based on computed similarity between users (or items). This requires heavy computations that distributed or decentralized architectures can scale down.

Nonetheless, they suffer drawbacks that trust reduces, by letting users define their own neighborhood. Trust-based approaches usually propagate trust in order to counter the trust network sparsity problem.

Social-based systems suffer less from social network sparsity, usually denser than trust network. They consider social networks in order to build user graphs. However most of them are global, therefore not P2P compliant.

The table 1 sums up approaches in this state of the art. It is divided in four parts, one per section from this chapter. We describe the different approaches regarding four characteristics: the kind of graph used by the system; the system knowledge on ratings, users or items; the required computation loads; and the graph relations origin.

Items or users similarity graphs imply computed graphs (thus implicit relations) using items' or users' profiles. Those profiles may require local (items characteristics) or global (all ratings) knowledge.

Trust and social networks are imported from existing user defined graphs (thus explicit relations). They may contain propagated relations using extended-local knowledge. That means that the system knows neighborhood up to a certain depth in order to propagate trust or social relations.

Similarity graphs require heavy computation since similarity is computed between all possible pairs of users or items. Likewise, matrix factorisation and homomorphic encryption require heavy computation resources.

Social networks contain social relations explicitly defined by users. We claim for a new social-based approach that does not need social / trust propagation. New social relations are not inferred without the users' consent.

Approach	Graph	Knowledge	Computation	Relations
TextBasedCB	items similarity	local	heavy	implicit
UserBasedCF	users similarity	global	heavy	implicit
ItemBasedCF	items similarity	global	heavy	implicit
TidalTrust	trust network	extended-local	light	propagated
MoleTrust	trust network	extended-local	light	propagated
RandomWalk	trust network	local	light	explicit
TrustWalker	trust & items similarity	global	heavy	explicit
SocialMF	social network	global	heavy	propagated
Hoens et al.	social network	local	heavy	explicit

Table 1: State of the art on recommender systems

Moreover, ratings are critical information regarding users preferences and behaviour. In order to provide a privacy policy, they should not be shared with anyone. Even when sharing data with trusted users, we think these data should not be easily identifiable by other users.

In this thesis, we introduce a recommender system propagating locally ratings through social relations. Our approach does not create new relations and shares data (*i. e.* ratings) only between direct friends. As it uses ratings aggregation, a final user does not know where ratings come from, following our privacy concern.

3

DEFINITIONS / EXAMPLE

Truth is not determined by majority vote.
— Doug Gwyn

Contents

3.1 Graph theory definitions 30
  3.1.1 Graphs 30
  3.1.2 Simple graphs 31
  3.1.3 Digraphs 32
  3.1.4 Bipartite graphs 32
  3.1.5 Weighted graphs 33
  3.1.6 Other graphs 34
3.2 Vocabulary 35
  3.2.1 Social network 35
  3.2.2 Trust network 36
  3.2.3 Similarity network 36
  3.2.4 Ratings 37
3.3 Example 37
3.4 Conclusion 38

Figures

Figure 1 Graph example 30
Figure 2 Simple graph example 31
Figure 3 Digraph example 32
Figure 4 Bipartite graph example 33
Figure 5 Weighted graph example 34
Figure 6 Other examples 34
  (a) Weighted digraph example 34
  (b) Bipartite digraph example 34
Figure 7 Social, trust and similarity networks and ratings example 38
  (a) Actors' ratings on item i_0 38
  (b) Social network 38
  (c) Trust network 38
  (d) Similarity network 38

This thesis is structured around several algorithms that need specific terminology. The section 3.1 provides some definitions needed for a good understanding on graph theory. The specific vocabulary used in this thesis is then described in section 3.2. This vocabulary is illustrated with a simple yet adequate example, provided in section 3.3. This example will be run with our algorithms in the entire chapter 4.

3.1 GRAPH THEORY DEFINITIONS

A graph is composed of a set of vertices and a set of edges linking the vertices. It is used to depict relationships (edges) between entities (vertices or nodes). Graphs have been widely used to model data in various situations. Different classes of graphs have been defined, for example directed or undirected, weighted or unweighted.

In this thesis, we use graphs to model social networks and ratings. We focus on undirected/directed, unweighted/weighted and bipartite graphs.

3.1.1 *Graphs*

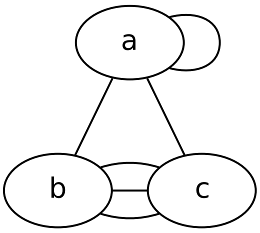


Figure 1: Graph example

Definition 4: Graph. *A graph* $G = (V, E)$ *is an ordered pair formed by a set of vertices* V *and a multiset of edges* E *depicting relationships between the vertices, where an edge is represented by a multiset of two vertices in* V.

Example 1. *Figure 1 depicts a graph with:*

- $A = \{a, b, c\}$
- $E = \{\{a, a\}, \{a, b\}, \{a, c\}, \{b, c\}, \{b, c\}, \{b, c\}\}$

Definition 5: Neighbourhood. *The* neighbourhood *of a vertex* v *in a graph* $G = (V, E)$, *denoted* $\Gamma_{\mathcal{G}}(v)$, *is the set of vertices directly connected to* v*:*

$$\Gamma_{\mathcal{G}}(v) = \{v' \in V \mid \{v, v'\} \in E\}$$

Example 2. *In figure 1:*

- $\Gamma_{\mathcal{G}}(a) = \{a, b, c\}$

- $\Gamma_{\mathcal{G}}(b) = \{a, c, c, c\}$
- $\Gamma_{\mathcal{G}}(c) = \{a, b, b, b\}$

Definition 6: Degree. *The* degree *of a vertex* v *in a graph* G *is* $|\Gamma_{\mathcal{G}}(v)|$.

Definition 7: Loop. *A loop is an edge that connects a vertex to itself.*

Example 3. *In figure 1, there is a loop on the vertex* a.

Definition 8: Cycle. *A cycle is a closed path,* i. e. *a path started from one vertex and ending on the same vertex, through other vertices.*

Example 4. *In figure 1, there are four cycles:*

- $(\{a, b\}, \{b, c\}, \{a, c\})$
- $(\{b, c\}, \{b, c\})$, *which appears 3 times*

3.1.2 *Simple graphs*

As shown in figure 1, graphs may contains loops (an edge from a node to the same node) and multiple edges (multiple edges from one node to another node). A simple graph is a graph with constraints on edges: no loop nor multiple edges are allowed. Simple graphs may however contain cycles.

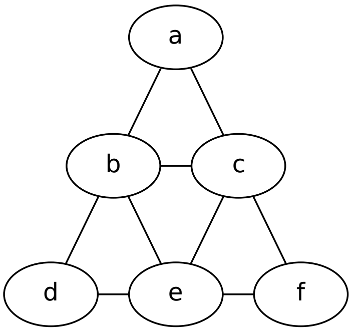


Figure 2: Simple graph example

Definition 9: Simple graph. *A simple graph* $G = (V, E)$ *is a graph such that* E *is a strict set and each edge in* E *is also a strict set,* i. e. *it starts from one vertex and ends in another:*

$$e = \{v, v'\} \in E \Leftrightarrow (v, v') \in A^2 \ \wedge \ v \neq v' \ \wedge \ \nexists e' \in E \mid e = e'$$

Example 5. *Figure 2 depicts a simple graph with the following set of vertices* $V = \{a, b, c, d, e, f\}$. *The set* $\{a, b\}$ *denotes an edge of the graph while* $\{a, e\}$ *does not.*

Since simple graphs are graphs, the neighbourhood and degree definitions described in section 3.1.1 are still valid.

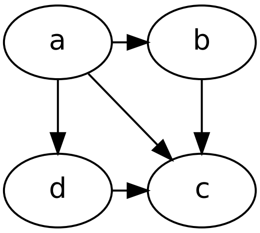


Figure 3: Digraph example

3.1.3 *Digraphs*

Directed graphs are called *digraphs* and consider directed relations between vertices.

Definition 10: Digraph. *A digraph* $D = (V, A)$ *is an ordered pair formed by a set of vertices* V *and a set of arcs* A *representing relationships between the vertices, where an arc is an ordered pair of two different vertices in* V*.*

Example 6. *Figure 3 depicts a digraph with the set of vertices* $\{a, b, c, d\}$*. The ordered pair* (a, b) *represents an arc of the graph from* a *to* b*, while* (b, a) *does not.*

Definition 11: Out-neighbourhood. *The* out-neighbourhood *of a vertex* v *in a digraph* $D = (V, A)$*, denoted* $\Gamma^{\rightarrow}_{\mathcal{D}}(v)$*, is the set of vertices directly connected to* v *with an arc starting from* v*:*

$$\Gamma^{\rightarrow}_{\mathcal{D}}(v) = \{v' \in V \mid (v, v') \in A\}$$

Definition 12: In-neighbourhood. *The* in-neighbourhood *of a vertex* v *in a digraph* $D = (V, A)$*, denoted* $\Gamma^{\leftarrow}_{\mathcal{D}}(v)$*, is the set of vertices directly connected to* v *with an arc ending in* v*:*

$$\Gamma^{\leftarrow}_{\mathcal{D}}(v) = \{v' \in V \mid (v', v) \in A\}$$

Example 7. *In figure 3:*

- $\Gamma^{\rightarrow}_{\mathcal{D}}(a) = \{b, c, d\}$ *and* $\Gamma^{\leftarrow}_{\mathcal{D}}(a) = \emptyset$
- $\Gamma^{\rightarrow}_{\mathcal{D}}(b) = \{c\}$ *and* $\Gamma^{\leftarrow}_{\mathcal{D}}(b) = \{a\}$
- $\Gamma^{\rightarrow}_{\mathcal{D}}(c) = \emptyset$ *and* $\Gamma^{\leftarrow}_{\mathcal{D}}(c) = \{a, b, d\}$
- $\Gamma^{\rightarrow}_{\mathcal{D}}(d) = \{c\}$ *and* $\Gamma^{\leftarrow}_{\mathcal{D}}(d) = \{a\}$

3.1.4 *Bipartite graphs*

A bipartite graph is a graph whose vertices can be divided into two disjoints sets U and V such that every edge connects a vertex from U to one in V. U and V usually do not depict the same kind of entity.

Definition 13: Bipartite graph. *A bipartite graph* $G = (U, V, E)$ *is an ordered tuple formed by two independent sets of vertices* U *and* V *and a set*

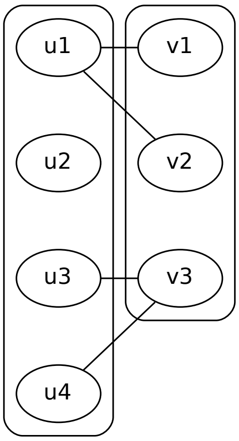


Figure 4: Bipartite graph example

of edges E *representing relationships between vertices from* U *and vertices from* V*, where an edge is represented by a set of a vertex in* U *and a vertex in* V*.*

Property 1. *In a bipartite graph* $G = (U, V, E)$*, nodes from the set* U *(respectively* V*) are linked to nodes from the set* V *(respectively* U*):*

$$\forall u \in U, \Gamma_{\mathcal{G}}(u) \subset V$$

$$\forall v \in V, \Gamma_{\mathcal{G}}(v) \subset U$$

Example 8. *Figure 4 depicts a bipartite graph* $G = (U, V, E)$ *with:*

- $U = \{u1, u2, u3, u4\}$
- $V = \{v1, v2, v3\}$
- $E = \{\{u1, v1\}, \{u1, v2\}, \{u3, v3\}, \{u4, v3\}\}$

The neighbourhood and degree concepts defined in section 3.1.1 are valid with bipartite graphs.

3.1.5 *Weighted graphs*

A weighted graph is a graph whose edges are weighted, *i.e.* a weight is associated with each edge.

Definition 14: Weighted graph. *A graph* $G = (V, E)$ *is weighted if and only if each edge* $e \in E$ *has a weight* $W(e) \in \mathbb{R}$*.*

Example 9. *Figure 5 depicts a weighted graph* $G = (V, E)$ *with:*

- $V = \{a, b, c, d\}$
- $E = \{\{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{c, d\}\}$
- $W(\{a, b\}) = 1$, $W(\{a, c\}) = 0.5$, $W(\{a, d\}) = 0$, $W(\{b, c\}) = 0.5$, $W(\{c, d\}) = 1$

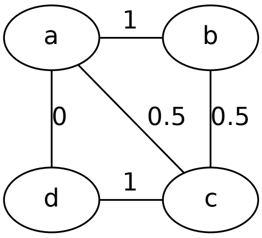


Figure 5: Weighted graph example

Definition 15: Positive neighbourhood. *The* positive neighbourhood *of a vertex v in a graph $G = (V, E)$, denoted $\Gamma^+_{\mathcal{G}}(v)$, is the set of vertices directly connected to v by an edge associated with a positive weight:*

$$\Gamma^+_{\mathcal{G}}(v) = \{v' \in \Gamma_{\mathcal{G}}(v) \mid W(\{v, v'\}) > 0\}$$

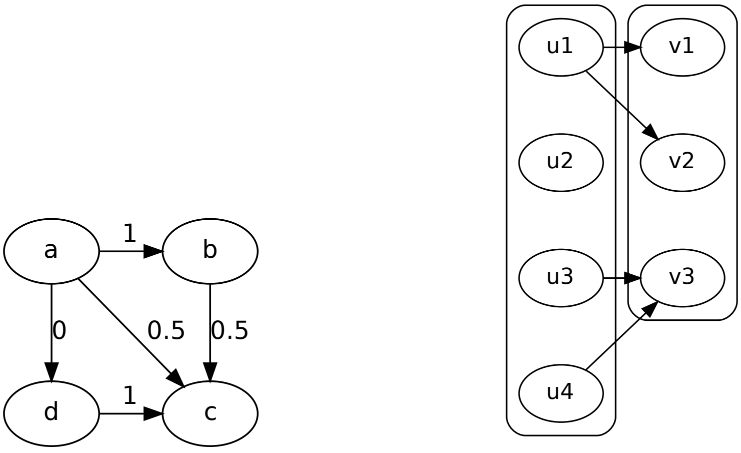
Example 10. *In figure 5,* $\Gamma_{\mathcal{G}}(a) = \{b, c, d\}$ *whereas* $\Gamma^+_{\mathcal{G}}(a) = \{b, c\}$ *since* $W(\{a, d\}) = 0$.

Definition 16: Positive degree. *The* positive degree *of a vertex v in a graph G is* $|\Gamma^+_{\mathcal{G}}(v)|$.

3.1.6 *Other graphs*

We have seen several definitions of graphs in the previous sections. A graph can be simple or not, directed or not, bipartite or not, weighted or not. However those definitions are not independent, a graph can be a mix of those.

In order to illustrate this, we show a weighted digraph and a bipartite digraph in figure 6.



(a) Weighted digraph example

(b) Bipartite digraph example

Figure 6: Other examples

Definition 17: Weighted digraph. *A weighted digraph* $D = (V, A)$ *is denoted as a regular digraph with one exception, weights are included in the arc triples. An arc is a triple associating two vertices from* V *and a weight.*

Example 11. *Figure 6a depicts a weighted digraph* $D = (V, A)$ *with:*

- $V = \{a, b, c, d\}$
- $A = \{(a, b, 1), (a, c, 0.5), (a, d, 0), (b, c, 0.5), (d, c, 1)\}$

Definition 18: Positive out-neighbourhood. *In a weighted digraph* $D = (V, A)$*, the* positive out-neighbourhood *of a vertex* v*, denoted* $\Gamma_{\mathcal{D}}^{+\rightarrow}(v)$*, is the set of vertices directly connected to* v *with an outgoing arc associated with a positive weight:*

$$\Gamma_{\mathcal{D}}^{+\rightarrow}(v) = \Gamma_{\mathcal{D}}^{\rightarrow}(v) \cap \Gamma_{\mathcal{D}}^{+}(v)$$

Example 12. *In figure 6a,* $\Gamma_{\mathcal{D}}^{+\rightarrow}(a) = \{b, c\}$ *since* $W(\{a, d\}) = 0$.

Definition 19: Bipartite digraph. *A bipartite digraph* $D = (U, V, A)$ *is denoted as a regular bipartite graph with arcs instead of edges.*

Example 13. *Figure 6b depicts a bipartite digraph* $D = (U, V, A)$ *with:*

- $U = \{u1, u2, u3, u4\}$
- $V = \{v1, v2, v3\}$
- $A = \{(u1, v1), (u1, v2), (u3, v3), (u4, v3)\}$

3.2 VOCABULARY

Given the graph definitions in the previous section, we specify some vocabulary used in this thesis that eases understanding. This vocabulary is illustrated with some examples in this section and a global example in section 3.3, figure 7, page 38.

3.2.1 *Social network*

A social network is a simple graph as defined in definition 9, section 3.1.2.

Definition 20: Social network. *A social network is a simple graph* $S_{\mathcal{G}} = (A, F)$*, with* A *a set of actors and* F *a set of social relations between them. A social relation is an undirected link between two actors in the social network. It is represented by a set of two actors* $\{a, a'\}$*, with* $(a, a') \in A^2$.

Remark 1 (Social network). We limit our definition to social networks having undirected links between actors, like Facebook. Social networks with directed links like Twitter are not considered.

Definition 21: Actor. *An actor refers to any social entity in a social network [WF94]* $a \in A$.

Remark 2 (Actor). An actor is not necessarily a user, it can also be any virtual entity connected to the network, such as a group of users or a company.

Definition 22: Friend. *A friend* f *of an actor* a *is an actor directly connected to* a *in the social network,* i. e. *a social relation exists between* a *and* f, i. e. $f \in \Gamma_{\mathcal{S}_\mathcal{G}}(a)$. *The set* F_a *of friends of* a *is the neighbourhood of* a*:*

$$F_a = \Gamma_{\mathcal{S}_\mathcal{G}}(a)$$

Remark 3 (Friend). We do not consider only friendship relations, but any kind of undirected links in a social network. However for clarity reason we call "friends" actors connected in the social network. The most important is that two "friends" accept to share their profiles.

3.2.2 *Trust network*

A trust network is a simple weighted digraph, *c.f.* definitions 9, 10 and 14. Trust relations are weighted and oriented relationships between actors ranging from 0 (lowest trust) to 1 (full trust). The greater the trust value from an actor a to an actor f, the more a trusts f's scoring, and then the more f's preferred items are valuable for a.

The trust relation from an actor a to an actor f is noted $t_{a,f}$. It is is neither symmetric nor transitive. Zero or one trust value $t_{a,f}$ can be associated with each $(a, f) \in A^2$.

Definition 23: Trust network. *A trust network is a simple weighted digraph* $T_\mathcal{D} = (A, T_\mathcal{A})$, *with* A *a set of actors and* $T_\mathcal{A}$ *a set of trust relations between them. A trust relation is an ordered triple of two actors and a trust value between them:* $(a, f, t_{a,f})$, *with* $(a, f) \in A^2$ *and* $t_{a,f} \in [0, 1]$.

3.2.3 *Similarity network*

A similarity network is a simple weighted graph, *c.f.* definitions 9 and 14. Similarity relations are weighted relationships between actors denoting a rating behaviour correlation between them. Weights rank from 0 (uncorrelated) to 1 (high similarity). They are computed through a correlation coefficient, as explained in section 2.2.1, eq.6, page 17. This coefficient is symmetric, hence an undirected graph. In this thesis we do not consider negative similarity between actors. The greater the similarity value between actor a and actor a', the more they tend to provide similar ratings.

The similarity relation between an actor a and an actor a' is noted $\rho_{a,a'}$. This relation is symmetric but not transitive. Zero or one similarity value $\rho_{a,a'}$ can be associated with each $(a, a') \in A^2$, with $\rho_{a',a} = \rho_{a,a'}$. Let $P_\mathcal{A}$ be the set of similarity tuples $(\{a, a'\}, \rho_{a,a'})$ with $(a, a') \in A^2$ and $\rho_{a,a'} \in [0, 1]$.

Definition 24: Similarity network. *A similarity network is a simple weighted graph* $P_{\mathcal{G}} = (A, P_{\mathcal{A}})$*, with* A *a set of actors and* $P_{\mathcal{A}}$ *a set of similarity relations between them. A similarity relation is a tuple of two actors and a similarity value between them:* $(\{a, a'\}, \rho_{a,a'})$*, with* $(a, a') \in A^2$ *and* $\rho_{a,a'} \in [0, 1]$.

Trust and similarity networks are similar kinds of graphs, representing different weights between actors: trust or similarity. However the first one is oriented while the second is not.

3.2.4 *Ratings*

A ratings graph is a simple weighted bipartite digraph, *c.f.* definitions 9, 10, 13 and 14. Ratings are set by actors on items. They are real values between 0 (the actor does not like the item) and 1 (the actor likes the item).

The rating given by an actor a to an item i is noted $r_{a,i}$. Zero or one rating $r_{a,i}$ can be associated with each $(a, i) \in A \times I$.

Definition 25: Rating network. *A rating network is a simple weighted bipartite digraph* $R_{\mathcal{D}} = (A, I, R_{\mathcal{A}})$ *between actors and items, with* A *the set of actors,* I *the set of items and* $R_{\mathcal{A}}$ *the set of ratings triple. A rating triple is composed of an actor, an item and a rating value between them:* $(a, i, r_{a,i})$ *with* $a \in A$, $i \in I$ *and* $r_{a,i} \in [0, 1]$.

Definition 26: Undefined rating. *Rating* $r_{a,i}$ *is undefined and noted* $\perp$ *if and only if actor* a *has not rated item* i*:*

$$\nexists (a, i) \in A \times I \mid (a, i, r_{a,i}) \in R_{\mathcal{A}} \Leftrightarrow r_{a,i} = \perp$$

Definition 27: Raters. *The set of raters* A_i *is the set of actors that have rated the item* i*:*

$$A_i = \{a \in A \mid \exists (a, i, r_{a,i}) \in R_{\mathcal{A}}\}$$

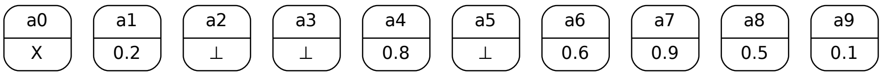
Definition 28: Friends raters. *The set of friends raters* $\mathcal{F}_{a,i}$ *of an actor* a *and an item* i *is the set of* a*'s friends that have rated item* i*:*

$$\mathcal{F}_{a,i} = F_a \cap A_i$$

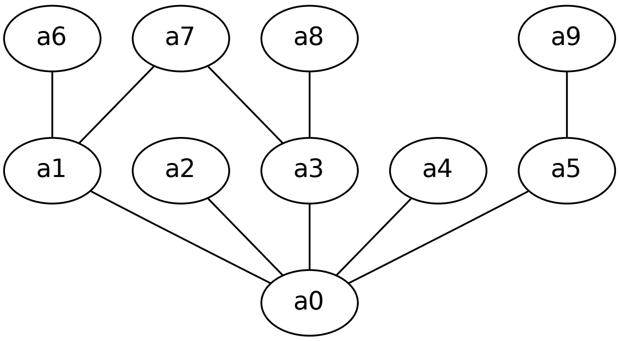
3.3 EXAMPLE

We illustrate these definitions with a simple, yet adequate, example shown in figure 7. Let us consider ten actors $\{a_0, a_1, \cdots, a_9\}$ and only one item i_0. The objective is to predict a rating from actor a_0 on item i_0, denoted X. For the sake of readability, we only show the networks, social, trust and similarity, starting directly or indirectly from our main actor a_0. Figure 7a shows ratings by actors on item i_0. a_1's rating is 0.2, a_2 and a_3 have not rated item i_0 yet, a_4's rating is 0.8, etc.

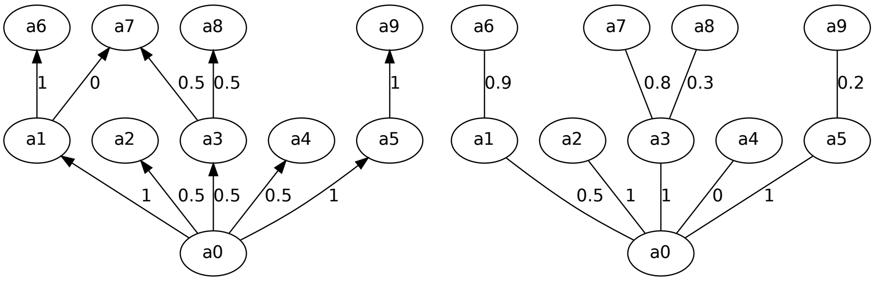
In our example:


a0
X
a1
0.2
a2
⊥
a3
⊥
a4
0.8
a5
⊥
a6
0.6
a7
0.9
a8
0.5
a9
0.1

(a) Actors' ratings on item i_0


a6
a7
a8
a9
a1
a2
a3
a4
a5
a0

(b) Social network


a6
a7
a8
a9
1
0
0.5
0.5
1
a1
a2
a3
a4
a5
1
0.5
0.5
0.5
1
a0
a6
a7
a8
a9
0.9
0.8
0.3
0.2
a1
a2
a3
a4
a5
0.5
1
1
0
1
a0

(c) Trust network (d) Similarity network

Figure 7: Social, trust and similarity networks and ratings example

- $A = \{a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9\}$
- $I = \{i_0\}$
- $T_{\mathcal{A}} = \{(a_0, a_1, 1), (a_0, a_2, 0.5), \cdots, (a_5, a_9, 1)\}$
- $P_{\mathcal{A}} = \{(\{a_0, a_1\}, 0.5), (\{a_0, a_2\}, 1), \cdots, (\{a_5, a_9\}, 0.2)\}$
- $R_{\mathcal{A}} = \{(a_1, i_0, 0.2), (a_4, i_0, 0.8), \cdots, (a_9, i_0, 0.1)\}$
- $F_{a_1} = \{a_0, a_6, a_7\}$, $F_{a_5} = \{a_0, a_9\}$, etc.
- $A_{i_0} = \{a_1, a_4, a_6, a_7, a_8, a_9\}$
- $\mathcal{F}_{a_0,i_0} = \{a_1, a_4\}$, $\mathcal{F}_{a_3,i_0} = \{a_7, a_8\}$, etc.

3.4 CONCLUSION

In this chapter, we have introduced definitions needed in the following of this thesis, based on graph theory. Using those definitions, we have defined some vocabulary such as social, trust, similarity and ratings networks.

Finally, we have specified an example containing all previous definitions: a social network between ten actors, trust and similarity between them and ratings on one item. This example illustrates our algorithms in chapter 4.

Part III

CONTRIBUTIONS

4

OUR APPROACH: SOCIAL SCORING

I do not know what I may appear to the world; but to myself I seem to have been only like a boy playing on the sea-shore, and diverting myself in now and then finding a smoother pebble or a prettier shell than ordinary, whilst the great ocean of truth lay all undiscovered before me.
— *Isaac Newton*

Contents

4.1 Social scoring 43
  4.1.1 Score propagation 43
  4.1.2 Trust 45
  4.1.3 Correlation 46
4.2 Confidence 47
  4.2.1 Confidence coefficients 48
  4.2.2 Confidence aggregation 52
  4.2.3 Confidence propagation 53
4.3 Default Score 55
  4.3.1 Computation of a default score 55
  4.3.2 Required knowledge for a default score . . 57
  4.3.3 Confidence on a default score 58
4.4 CoTCoDepth Social Scoring 59
  4.4.1 Definition 59
  4.4.2 Example 59
4.5 Conclusion 61

Figures

Figure 8 k−Depth Social Scoring Example 45
Figure 9 Trust Example 45
Figure 10 Correlation Example 46
Figure 11 Size confidence depending on number of friends' scores 49
Figure 12 Freshness confidence depending on scores age with different λ 51
Figure 13 Confidence Example 54
Figure 14 CoTCoDepth Example 60

In this thesis, we focus on a recommender system that propagates rating predictions, compatible with peer to peer architectures. We also consider users privacy by limiting profile disclosure.

The state of the art, chapter 2, shows that most trust-based recommender systems propagate trust to counter sparsity. More generally, collaborative filtering approaches use global knowledge on ratings in order to compute similarity. Few systems offer data privacy to users and distributed recommendation.

In the previous chapter, we have determined the vocabulary used in this chapter to define our approach: social, trust and similarity networks as well as ratings (section 3.2 page 35). Followed by our main example (section 3.3 page 37).

Our privacy assumptions state that users should define whom they want to share their data with. In the following, we estimate that users want to share their data with their friends. We define a score as either a rating if a user has one, or an aggregation of friends' scores. Our social scorer propagates scores through users' social network. It offers users the possibility to define a friend as trustworthy or not [1]. Respecting privacy and unlike traditional approaches, we provide similarity between friends using local knowledge. Peers, *i. e.* actors, host locally their own private profiles and may share them on demand with their direct neighbors, *i. e.* friends.

In order to provide accurate recommendation, we propose to take into account the following criteria: neighborhood selection, neighbors weight, scores weight and sparsity.

Actors neighborhood selection is the first step. We focus on social relations in order to select neighbors and how to propagate scores throughout them. However, actors should be allowed to explicitly defined whom they want recommendation from, thanks to trust coefficients. This requires some efforts from end users, therefore the system should provide automatic selection of similar neighbors using similarity coefficients.

In addition, scores returned by neighbors may also be considered differently depending on how they ave been computed: some are more reliable than others. Confidence on scores have to be considered.

Finally, it is essential to counter sparsity with decentralized architectures constraints. When classical prediction fails, we need to artificially return scores, *aka.* default scores.

In section 4.1, we present the score propagation in the actors' social network. We introduce two coefficients between actors aiming at improving recommendation accuracy. Trust allows actors to explicitly weight friends' ratings and correlation modulates friends' ratings by computing similarity between friends.

---

[1] We remind that in this context, trust means that a user believes on the usefulness of the recommendation of a trusted user.

Then, we introduce confidence in section 4.2, a coefficient from the system on scores. Confidence values adjust scores importance by highlighting which scores seem to be accurate. The confidence is provided to the final user to indicate the system belief on the recommendation result accuracy. Several factors impact on the confidence computation, such as actors' distance, actors' weight and actors' confidence, number of friends, similarity of friends' scores and freshness of the recommendation.

Section 4.3 introduces default scoring strategies in order to remedy sparsity problems. Those strategies use local knowledge or anonymous global knowledge in order to return a score even when friends cannot help predict one. The purely local strategy is fully compatible with our hypothesis and is based on actor's ratings. The anonymous one requires some global information on item's ratings, but no indication on who made those ratings.

We finally present our CoTCoDepth scorer in section 4.4. CoTCoDepth stands for Correlative and Trust-based with Confidence k-Depth social recommender system. It combines the different definitions from this chapter to propose an accurate recommendation with a valued confidence, solely based on actors' social network.

4.1 SOCIAL SCORING

In this section, we introduce our generic formula called "k−depth social scoring" (s_k) that forms the basis of our social scoring. This formula uses propagation and requests in the social network. It is based on trust between friends as well as correlation between friends profiles.

Our formula uses a coefficient ω that is defined in the following. It can be based on trust (*c.f.* section 4.1.2), similarity (*c.f.* section 4.1.3) or both.

4.1.1 *Score propagation*

In the following, we use a specific vocabulary to describe some actions in the system.

ASKING Asking a score denotes sending a request from a peer (an actor) to another peer (his/her friend) in order to get a score relative to an item from this friend.

REQUESTER A requester is a peer asking for a score, at each step of the propagation.

ORIGINAL REQUESTER The original requester is the peer asking for a score on the first propagation, the one who receives the final recommendation.

$k-$depth social scoring gets actor's rating if it exists. If not, it asks the actor's friends to provide their ratings (if any) or to predict their scores, using their friends' ratings and so on, to depth k. If no one returns a score, the final score is unpredicted ($\perp$). If a score is unpredicted, the system cannot provide any recommendation for this specific item at this point.

The coefficient $\omega_{a,f}$ between actors a and f weights the mean of friend's scores. It will be specified in the following sections. s_k is the score computed at the k^{th} step of propagation.

Definition 29:. *Let* $\mathcal{F}^k_{a,i,\omega}$ *be the set of* a*'s friends* f *where* $s_k(f,i)$ *is defined at step* k *and* $\omega_{a,f}$ *is not null.*

$$\mathcal{F}^k_{a,i,\omega} = \{f \in F_a \mid s_k(f,i) \neq \perp \wedge \omega_{a,f} \neq 0\} \tag{12}$$

The $k-$depth social scoring is then defined as in eq. 13[2]. This formula is an adaptation to our constraints of the one described in section 2.2, eq. 4, page 16.

$$s_k(a,i) = \begin{cases} r_{a,i} & \text{if } \exists r_{a,i} \\ \dfrac{\sum_{f \in \mathcal{F}^{k-1}_{a,i,\omega}} \omega_{a,f} \times s_{k-1}(f,i)}{\sum_{f \in \mathcal{F}^{k-1}_{a,i,\omega}} \omega_{a,f}} & \text{if } \nexists r_{a,i} \wedge \mathcal{F}^{k-1}_{a,i,\omega} \neq \emptyset \\ \perp & \text{otherwise} \end{cases} \tag{13}$$

By definition, $s_0(a,i)$ is the rating $r_{a,i}$ set by the actor a on item i if exists, $\perp$ otherwise:

$$s_0(a,i) = \begin{cases} r_{a,i} & \text{if } \exists r_{a,i} \\ \perp & \text{otherwise} \end{cases} \tag{14}$$

Figure 8 runs the motivating example with $\omega_{a,f} = 1$ and $k = 2$. a_0 asks all his/her friends their scores for the item. a_1 has already a rating and returns immediately 0.2 without asking a_6's score (symbolized by a black rectangle). a_2, having no rating nor friend, returns $\perp$. a_3 has no rating but two friends, a_7 and a_8. The former computes a mean using the two latter ratings and returns it (0.7). a_4's rating is transmitted and a_5 returns a_9's rating. With this $k-$depth social scoring, a_3 and a_5 help a_0 to compute the score through their friends and then participate in the score computation.

The $k-$depth social scoring propagates scores only between immediate friends. Scores are aggregated before being transmitted to the requester, then scores are propagated in the network without interfering with existing relations.

The following sections describe different coefficients ω used in our approach.

---

[2] For clarity, we use a simplified version of the formula. The complete version is described in appendix A.

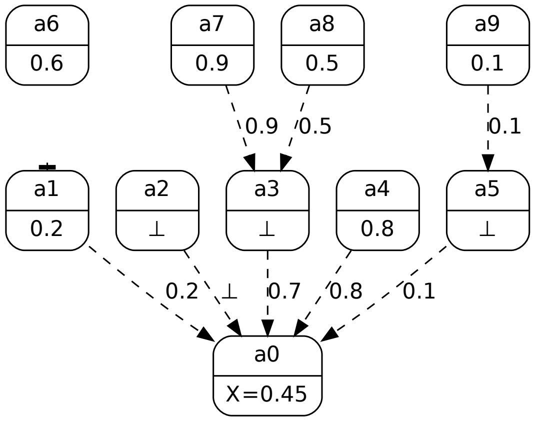
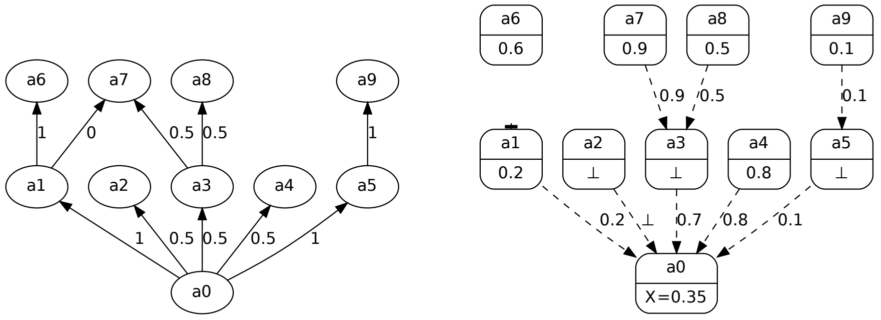

a6
0.6
a7
0.9
a8
0.5
a9
0.1
0.9
0.5
0.1
a1
0.2
a2
⊥
a3
⊥
a4
0.8
a5
⊥
0.2
⊥
0.7
0.8
0.1
a0
X=0.45

Figure 8: k−Depth Social Scoring Example

4.1.2 *Trust*

Trust values are explicitly defined by actors between friends. As defined in section 3.2.2, trust relations are weighted and oriented relationships between friends ranging from 0 (lowest trust) to 1 (full trust). This allows our approach to be compatible with both trust networks and social networks.

If scores are propagated in a trust network or in a social network containing trust values, trust values are considered to weight relations: $\omega_{a,f} = t_{a,f}$ in eq.13. If scores are propagated in a social network that cannot manage trust values, we set trust to 1 between all friends, cancelling trust coefficient: $\omega_{a,f} = 1$ in eq.13. If scores are propagated in a social network containing some relations with trust value and some other without, a default trust value specific to the social network should be explicitly defined. Typically it can be 0 (no one is considered but whitelisted trusted actors) or 1 (everyone is considered but blacklisted untrusted actors).


a6
a7
a8
a9
1
0
0.5
0.5
1
a1
a2
a3
a4
a5
1
0.5
0.5
0.5
1
a0
a6
0.6
a7
0.9
a8
0.5
a9
0.1
0.9
0.5
0.1
a1
0.2
a2
⊥
a3
⊥
a4
0.8
a5
⊥
0.2
⊥
0.7
0.8
0.1
a0
X=0.35

(a) Trust network

(b) Score propagation

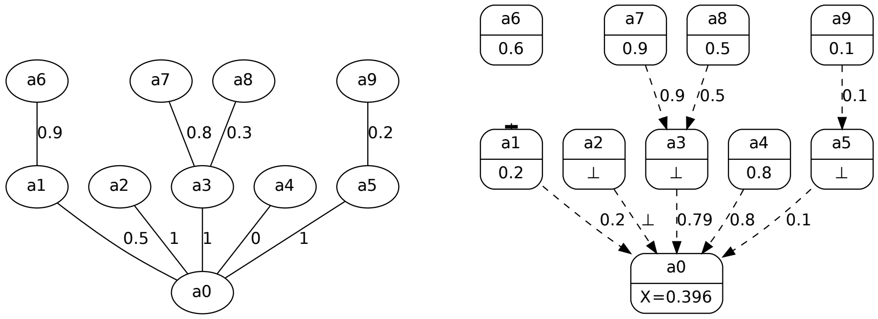
Figure 9: Trust Example

Score propagation in figure 9 is the same as previously (*c.f.* figure 8), except that scores are weighted by trust. In our example, the predicted score for a_0 is then 0.35 since a_0 trusts less a_3 and a_4 who have high ratings for i_0. Since actors define trust explicitly, this allows them to refine the importance of some friends.

4.1.3 *Correlation*

Trust relations are explicitly defined by actors and therefore subjective. We want to detect automatically friends that are more likely to provide accurate recommendations. We thereby use similarity, estimated thanks to a correlation coefficient.

Unlike global approaches, correlation is not computed between all actors to build a global graph, but only between direct friends. It modulates the existing social graph without adding new links. This coefficient is a classic Pearson's correlation coefficient, as described in section 2.2.1 (eq. 6 page 17), denoted by ρ.



(a) Similarity network

(b) Score propagation

Figure 10: Correlation Example

As shown in figure 10, correlation changes the way actors deal with scores. It varies how one friend infers in the score computation among other friends. a_7 and a_8 share their ratings as before, but since a_3 is more similar to a_7, a_7's score has a higher weight. a_0 does not take into account a_4's score since they share no similarity.

Thanks to the correlation, the scorer takes into account similarity between friends to compute scores, friends with similar tastes are "promoted" during the recommendation.

To compute $\omega_{a,f} = \rho_{a,f}$ between two friends a and f, only items rated by both friends are used. However with sparse social networks, such as datasets described in section 5.2 (page 67), friends and similar actors are usually disjoint. In order to take into account this sparsity, we define an extended version of our local similarity in section 6.2, page 86.

If two friends still have no item in common, a default value is returned[3]: 0.5, the average between no similarity (0) and full similarity (1). This allows the system to get recommendation from all friends.

4.2 CONFIDENCE

In the previous sections, actors are weighted by trust and/or similarity but each score has the same weight. We think that all scores should not be treated equally during the aggregation and propagation.

Indeed, the system cannot be as confident on a score computed from only one rating given by one distant friend as on a score computed from many friends giving the same recommendation.

Currently, an actor returns a score to his/her friend with his/her own weight, even if this score has been computed using friends with a completely different weight. In the previous example, a_5 has only one friend a_9 and since the mean is normalized by the sum of all similarities, the similarity between a_5 and a_9 does not change anything. Then, a_9's rating is as important as a_7's and a_8's gathered through a_3, despite a smaller similarity.

We estimate conditions that should be followed to provide accurate predictions. Indeed, a prediction should be more accurate when:

1. friends provide scores they are confident on (*aka.* friends' confidence),
2. scores come from highly trusted or similar friends (*aka.* friends' weight),
3. an actor aggregates multiple friends' scores instead of a few (*aka.* scores set size),
4. friends' scores are equal instead of being highly different (*aka.* scores' variance),
5. scores are recent (*aka.* scores' freshness),
6. scores come from direct friends instead of friends of friends (*aka.* distance).

A score confidence is the belief of the system on the accuracy of its prediction. The higher the confidence, the higher the probability of the recommendation to be accurate, according to the system.

In the following sections, we define confidence coefficients that take into account these conditions, then aggregate those coefficients and finally propagate the aggregation in the network along with the score propagation.

---

[3] We want to take into account all actor's friends, with similar ones having more impact than others. This default similarity has been validated empirically after tests on different values.

4.2.1 *Confidence coefficients*

In order to favor scores respecting accuracy conditions, we have introduced confidence from actors on scores: $c \in [0, 1]$. 0 means that the score is likely to be not accurate at all, 1 means that the system is confident on the score accuracy. This coefficient is associated and transmitted with each score.

Confidence is computed from various coefficients in order to cope with the conditions described above.

4.2.1.1 *Friends confidence* ($c^{\mathcal{F}}$)

This coefficient copes with condition 1 described above. It is the mean of friends confidence on their scores, weighted by their ω coefficients.

With a given depth k, the friends confidence from actor a to i's score is :

$$c^{\mathcal{F}}_{a,i} = \frac{\sum_{f \in \mathcal{F}^k_{a,i,\omega}} \omega_{a,f} \times c_{f,i}}{\sum_{f \in \mathcal{F}^k_{a,i,\omega}} \omega_{a,f}}$$

The lower friends are confident on their scores, the lower $c^{\mathcal{F}}$. In the meantime, friends with high coefficients are more likely to influence this confidence.

However, if all friends have a high confidence on their scores but all ω between them and the actor are low, friends confidence will still be high.

4.2.1.2 *Weight confidence* (c^{ω})

This coefficient corresponds to condition 2 described above and cope with the friends confidence drawback describe previously. We consider that if all friends that provide a score have a low ω coefficient, the confidence should remain low. Therefore we define the weight confidence as the maximum of friends' weights ω. If at least one friend's weight is high, this coefficient will be high, otherwise it will remain low.

With a given depth k and a given weight ω, the weight confidence from actor a to i's score is :

$$c^{\omega}_{a,i} = \max_{f \in \mathcal{F}^k_{a,i,\omega}} \omega_{a,f}$$

This coefficient takes into account cases where an actor has many friends highly confident on their recommendations, but where the links between the actor and his/her friends have low weights. If at least one weight is high, the associated confidence will impact more the friends confidence coefficient, which handles cases with mixed high and low weights.

4.2.1.3 *Size confidence* (c^{size})

This confidence takes into account the number of friends' scores (condition 3). The more friends' scores, the higher the confidence on the score.

We have chosen a logistic function (the sigmoid, *c.f.* éq.15) to model that confidence: it is a monotonic increasing function, the initial growth (for positive values) is approximately exponential, followed by a slowing down until reaching the value 1 (*c.f.* figure 11).

$$sigmoid(x) = \frac{1}{1+e^{-x}} \tag{15}$$

In order to adapt the logistic function to our case, we define some properties:

- without score, this confidence has no sense,
- we set 0.5 as the lowest size confidence, *i.e.* the confidence if there is only one friend's score (flip-coin prediction),
- no need to have a lot of friends' scores to have high confidence[4].

Therefore, with a given depth k, the size confidence of actor a on i's score is:

$$c^{size}_{a,i} = sigmoid(|\mathcal{F}^{k}_{a,i,\omega}| - 1) \tag{16}$$

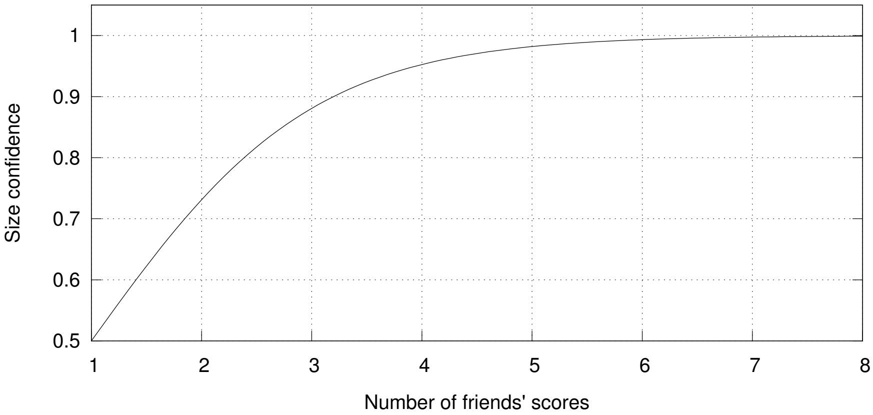


Figure 11: Size confidence depending on number of friends' scores

This confidence goes from 0.5 with only one friend's score to about 1 with 7 friends' scores or more.

---

[4] Our experimentations show that five friends are enough to provide good accuracy, therefore high confidence (*c.f.* section 6.5.2 page 93).

4.2.1.4 *Variance confidence* (c^σ)

This confidence takes into account the variance of friends' scores (condition 4). The higher the variance, *i. e.* the more different the friends' scores, the lower the confidence on the score. [JE09] also uses a variance computation in order to provide a confidence on the computed score to the final user. But our approach uses a weighted variance, taking into account scores' weights, and uses this variance during the score computation, not only at the end.

A weighted variance is computed on the friends' scores. With a given depth k, the variance confidence of actor a on i's score is:

$$c^\sigma_{a,i} = 1 - \frac{\sigma^2_{a,i}}{\sigma^2_{max}} \tag{17}$$

$$\sigma^2_{a,i} = \frac{\sum_{f \in \mathcal{F}^k_{a,i,\omega}} \omega_{a,f} \times (s_{k-1}(f,i) - \mu^*)^2}{\sum_{f \in \mathcal{F}^k_{a,i,\omega}} \omega_{a,f}} \tag{18}$$

μ^* is the actor's friends' scores weighted mean. σ^2_{max} is the maximum possible variance and is used to normalize the confidence. As stated by [JE09], $\sigma^2_{max} = \frac{Range^2}{4}$ for a dataset with a finite rating range denoted *Range*.

4.2.1.5 *Freshness confidence* (c^t)

This confidence aims at taking into account obsolescence of one's ratings (condition 5). It is specific to explicit ratings and does not consider computed scores. This coefficient is only considered when ratings are timestamped, as explained in section 4.2.2.1.

The freshness is function of the age of the rating: the older the less confident on a rating. We bound freshness to $]0.5, 1]$ with the following assumptions:

1. it starts from 1: the highest confidence is when the rating has just been made,
2. it remains greater than 0.5: an old explicit rating is still an explicit rating made by the actor, thus a valid rating[5].

These assumptions are generic but the freshness should be specific to items since some items ratings become obsolete faster than others. Therefore we define two parameters allowing us to tune the freshness according to the kind of recommended items:

- the half-life λ is the period of time after which the confidence lost about half its amplitude, *i. e.* equals 0.75 or so,
- the time unit $\mathcal{T}$, or scale, give the lifetime of a recommendation: minutes, days, months, etc.

In order to model the freshness function, we have also chosen a logistic function based on the *sigmoid* function defined in eq.15 page 49 (t is in $\mathcal{T}$ unit).

---

[5] A value lower than 0.5 would reduce the rating impact too much

$$f_\lambda(t) = \alpha_\lambda \times \mathit{sigmoid}(\lambda - t) + \beta_\lambda$$

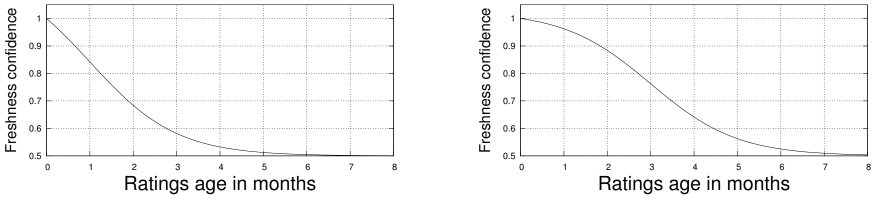
The freshness is function of the age of the rating and monotonically decreasing. To satisfy the conditions 1 and 2, we need to define α_λ and β_λ.

$$\begin{aligned}
\text{condition } 2 &\Leftrightarrow \lim_{t\to\infty} f_\lambda(t) = 0.5 \\
&\Leftrightarrow \alpha_\lambda \times \lim_{t\to\infty} \mathit{sigmoid}(\lambda - t) + \beta_\lambda = 0.5 \\
&\Leftrightarrow \beta_\lambda = 0.5 \\
\text{condition } 1 &\Leftrightarrow f_\lambda(0) = 1 \\
&\Leftrightarrow \alpha_\lambda \times \mathit{sigmoid}(\lambda) + 0.5 = 1 \\
&\Leftrightarrow \alpha_\lambda = \frac{1}{2 \times \mathit{sigmoid}(\lambda)}
\end{aligned}$$

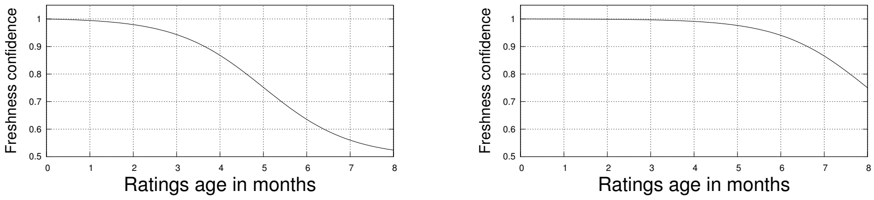
Therefore, c^t is defined as:

$$c^t_{a,i} = \frac{\mathit{sigmoid}(\lambda - t_{a,i})}{2 \times \mathit{sigmoid}(\lambda)} + 0.5 \tag{19}$$

The freshness confidence verifies all assumptions defined above. It remains greater than 0.5 but in the infinite. In figure 12 are shown some examples with different λ and a unit time $\mathcal{T}$ in months.



(a) $\lambda = 1$ (b) $\lambda = 3$



(c) $\lambda = 5$ (d) $\lambda = 8$

Figure 12: Freshness confidence depending on scores age with different λ

Since λ and $\mathcal{T}$ are dependant on the kind of recommended items, they should be defined either by the users or the items category. We can assume that a tweet will have a low λ coefficient and a time unit $\mathcal{T}$ in hours or days, whereas a movie will have higher λ and $\mathcal{T}$.

4.2.1.6 *Distance*

In order to cope with condition 6, we choose not to define another coefficient. At each score propagation, the actual distance between a peer and the original requester is known neither by the original requester nor by the peer. However the confidence should decrease at each hop in order to promote direct recommendations, given by direct friends, compared to indirect or distant recommendations.

This problem is solved by the confidence aggregation, described in the following section, where the confidence tends to be lower at each hop in the score propagation.

4.2.2 *Confidence aggregation*

Since only one coefficient is propagated in the network, each peer aggregates the confidence coefficients before transmission. We consider two different cases when an actor transmits a score with a confidence: either it is an explicit rating; either it is a computed score. Depending on the situation, the confidence is not computed in the same way.

4.2.2.1 *Rating confidence*

If an actor has rated an item, he/she did not ask any recommendation from his/her friends. Therefore most of the confidence coefficients are not computable (friends, weight, size, variance confidences). Only the freshness confidence is available if the rating is associated with a time.

Therefore if the rating time is available, actors' confidence on their own rating is the freshness confidence: $c_{a,i} = c^{t}_{a,i}$. Otherwise the confidence is 1, as we assume actors to be confident on their own ratings.

4.2.2.2 *Score confidence*

When an actor has not rated an item, the confidence on the calculated score is computed using the other coefficients: size, friends, weight and variance confidences.

If all coefficients are maximum (respectively minimum), then the aggregated confidence should be maximum (respectively minimum). But those coefficients are not independent from each others. The more friends return scores, the more friends, weight and variance confidences are reliable.

The size confidence should influence the aggregation specifically: a high size confidence implies that the other coefficients are reliable, so we should only use them; a low size confidence implies that the overall confidence should be low, since the other coefficients are not reliable.

Therefore the aggregated confidence c is proportional to the size confidence and to the mean of the friends, weight and variance confidences:

$$c_{a,i} = c_{a,i}^{size} * \frac{c_{a,i}^{\mathcal{F}} + c_{a,i}^{\omega} + c_{a,i}^{\sigma}}{3}$$

With a high size confidence (near 1), the overall confidence is mainly computed thanks to the friends, weight and variance confidence. With a low size confidence (near 0.5), the overall confidence is low, no matter the other coefficients. Then the size confidence is always the maximum of the overall confidence.

4.2.2.3 *Confidence formula*

The complete formula to compute confidence is then:

$$c_{a,i} = \begin{cases} c_{a,i}^{t} & \text{if } \exists r_{a,i} \\ c_{a,i}^{size} * \dfrac{c_{a,i}^{\mathcal{F}} + c_{a,i}^{\omega} + c_{a,i}^{\sigma}}{3} & \text{if } \nexists r_{a,i} \wedge \mathcal{F}_{a,i,\omega}^{k-1} \neq \emptyset \\ \perp & \text{otherwise} \end{cases} \tag{20}$$

In order to have the highest confidence, *i. e.* 1, one needs ideal conditions: a lot of identical recommendations coming from friends confident on their scores. Most of the time it will not be the case, therefore the confidence is usually less than 1. This tends to lower confidence at each propagation, the distance (condition 6) is processed at this point: each hop in the network will lower step by step the confidence. Scores coming from direct neighbours will have a higher confidence than scores coming from indirect neighbours.

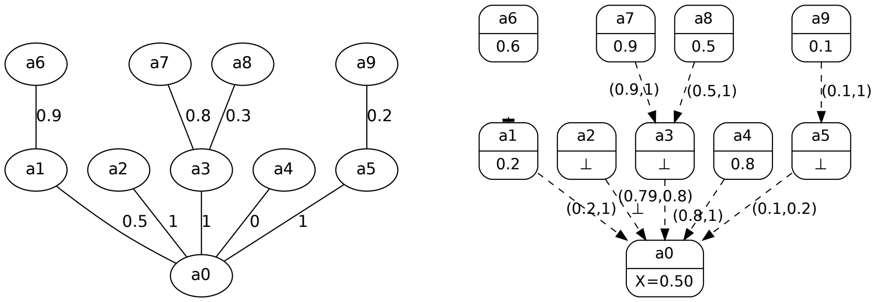
4.2.3 *Confidence propagation*

During the scores propagation, confidences are transmitted along with scores. Score computation uses only ω, except for the original requester. The original requester takes into account this confidence coefficient in the new score computation. The final coefficient (noted $\omega^{(c)}$), for a given item i, is then:

$$\omega_{a,f}^{(c)} = \omega_{a,f} \times c_{f,i}$$

To illustrate, the coefficient ω is based on similarity in figure 13. For clarity purposes, we only use friends confidence, defined in section 4.2.1.1 and weight confidence, defined in section 4.2.1.2.

a_1 and a_4 are fully confident on their own ratings and return 1 as scores confidence. a_2 does not return any score nor confidence.



(a) Similarity network (b) Score propagation

Figure 13: Confidence Example

a_5 computes a score using only one friend, therefore the friends confidence is 1 and the weight confidence is his/her friend's weight: 0.2. a_5's confidence is thus 0.2.

a_3 computes a score using a_7 and a_8's scores. Both a_7 and a_8 are fully confident on their own scores, so a_3's friends confidence is 1. a_3's weight confidence, being the maximum friends weight, is 0.8. Finally a_3's score confidence is:

$$c_{a_3,i_0} = c^{\mathcal{F}}_{a_3,i_0} \times c^{\omega}_{a_3,i_0} = 1 \times 0.8 = 0.8$$

Scores are propagated with their confidences to a_0, who takes them into account on the final score computation:

$$\begin{aligned} s_2(a_0,i_0) &= \frac{\sum_{f \in \mathcal{F}^1_{a_0,i_0,\omega}} \omega_{a_0,f} \times c_{f,i_0} \times s_{\mathcal{I}}(f,i_0)}{\sum_{f \in \mathcal{F}^1_{a_0,i_0,\omega}} \omega_{a_0,f} \times c_{f,i_0}} \\ &= \frac{(0.5 \times 1 \times 0.2) + (1 \times 0.8 \times 0.79) + (0 \times 1 \times 0.8) + (1 \times 0.2 \times 0.1)}{(0.5 \times 1) + (1 \times 0.8) + (0 \times 1) + (1 \times 0.2)} \\ &= 0.50 \end{aligned}$$

Then a_0 computes his/her final confidence on the score using friends and weight confidence:

$$\begin{aligned} c^{\mathcal{F}}_{a_0,i_0} &= \frac{(0.5 \times 1) + (1 \times 0.8) + (0 \times 1) + (1 \times 0.2)}{0.5 + 1 + 0 + 1} = 0.6 \\ c^{\omega}_{a_0,i_0} &= \max(0.5, 1, 0, 1) = 1 \\ c_{a_0,i_0} &= c^{\mathcal{F}}_{a_0,i_0} \times c^{\omega}_{a_0,i_0} = 0.6 \times 1 = 0.6 \end{aligned}$$

In this example, confidence reduces a_9's low rating impact on the prediction, since the latter shares low similarity with a_5.

Confidence locally reduces the weight of scores propagated from friends of friends. The more friends are related (regarding trust or similarity), the higher their confidence. In addition, confidence is provided to the final user as an accuracy indicator about the recommendation.

We have seen several techniques to propagate scores and enhanced accuracy. The last proposition concerns the improvement of coverage.

4.3 DEFAULT SCORE

In classical collaborative filtering approaches as well as in trust-based approaches, only ratings from the recommended item are used to compute a score. That means that an item without rating cannot be recommended. That also means that an item with few ratings has few chances to be recommended, especially when the actors who rated this item are not linked to many people in the actors graph. Due to the usual sparsity of existing datasets, many item scores cannot be computed since no friend has rated these items up to the given depth, which leads to a limited coverage. This is a well known problem in recommendation often named *cold start* problem.

Default scoring has been introduced to counter this problem, *c.f.* section 2.2.3 page 18. However among trust-based recommender systems presented in the state of the art, only TrustWalker [JE09] uses ratings from other items than the recommended one. TrustWalker computes similarity between items and may return a rating from a similar item as a default score when an actor in the trust network has not rated the item to recommend. This requires a global knowledge on items' ratings and who made them.

Despite this approach, trust-based recommender systems tend to propagate deeper in the network in order to counter this drawback, implying heavier traffic, longer calculation time and more knowledge disclosure thus lower privacy.

In this section, we propose default scores adapted to our decentralized architecture and/or to our privacy objectives. In this approach, an actor may return a default score if he/she has not be able to compute a score in another way.

This approach increases coverage by providing a score for almost every requester, without propagating deeper in the network.

In section 4.3.1, we first explain the default score computation and the probability to return a default score. We then model and describe required knowledge in our approaches and in [JE09]'s approach in section 4.3.2. We finally adapt the confidence computation with default scores in section 4.3.3.

4.3.1 *Computation of a default score*

We choose not to always return a default score when a rating is not computable. Section 4.3.1.1 defines the probability to return such a default score for a given actor. We propose two strategies to compute a score in section 4.3.1.2. The first one is purely local and the second one is global but requires only anonymous data, thus not compatible with decentralized architectures but respecting our privacy concerns.

4.3.1.1 *Default score probability*

We define $P_{default}$ as the probability to return a default score if an actor cannot compute a score. This adds randomness in the recommendation, which is usually considered as a good thing in order to recommend new items to a specific actor [AT05]. Limiting default score propagation with that probability also reduces the computational burden by not always returning a score. Finally, it minimizes noisy recommendations, *i.e.* recommendations not based on a specific item. For that purpose, this probability must remain low.[6]

Therefore, equation 13 (page 44) becomes:

$$s_k(a,i) = \begin{cases} r_{a,i} & \text{if } \exists r_{a,i} \\ \dfrac{\sum_{f \in \mathcal{F}^{k-1}_{a,i,\omega}} \omega_{a,f} \times s_{k-1}(f,i)}{\sum_{f \in \mathcal{F}^{k-1}_{a,i,\omega}} \omega_{a,f}} & \text{if } \nexists r_{a,i} \wedge \mathcal{F}^{k-1}_{a,i,\omega} \neq \emptyset \\ \mathit{default}(a,i) & \text{otherwise} \end{cases} \tag{21}$$

With *default*(a,i) defined with the following probabilities:

$$\begin{aligned} P(\mathit{default}(a,i) \neq \perp) &= P_{default} \\ P(\mathit{default}(a,i) = \perp) &= (1 - P_{default}) \end{aligned} \tag{22}$$

$P_{default} = 0.02$ means that 2 % of friends who cannot compute a score will return the default score instead of $\perp$. In our evaluation, lower probability handicaps coverage and higher probability penalizes both accuracy and prediction time.

4.3.1.2 *Default score value*

When *default*$(a,i) \neq \perp$, we define two different strategies to compute this default score. The first one is local and uses the actor profile. The second one is global but anonymous and uses the item profile.

The reader should note that computing items' similarity as TrustWalker does is not possible without knowing the detail of all ratings on items, thus requiring a global knowledge as explained in the following section.

ACTOR'S MEAN is the local one. The actor returns his/her own ratings mean as a default score. This allows the system to have an average of what this actor would have rated if he/she has known/used this item. This does not require more data than the profile of the actor returning the default score.

$$\mathit{default}(a,i) = \overline{r_a} \tag{23}$$

---

[6] In our evaluation, we have empirically set $P_{default} = 0.02$.

ITEM'S MEAN is the global but anonymous one. The actor returns the item's ratings mean as a default score. This allows the system to have a rating based on all actors on this item, therefore it is usually more accurate than the previous one. This requires to know more than just the actor's profile: it requires to know all ratings on this item and thus centralized architectures. However those ratings are anonymous.

$$\mathit{default}(a, i) = \overline{r_i} \tag{24}$$

4.3.2 *Required knowledge for a default score*

When an actor cannot compute a score for a specific item and needs to return a default score, this default score is computed thanks to ratings other than the rating from this actor on this item. We have seen in the previous section that the two strategies *actor's mean* and *item's mean* do not require to know the same kind of ratings.

We assume that actor a computes a default score on item i.

Definition 30: Ratings knowledge. *We define the sets* R, R_a *and* R_i *according to the following:*

- R *is the set of all ratings from actors on items:* $R = \{r_{a',i'} | a' \in A \wedge i' \in I\}$.
- R_i *is the set of ratings from actors on item* i*:* $R_i = \{r_{a',i} | a' \in A\}$.
- R_a *is the set of ratings from actor* a *on items:* $R_a = \{r_{a,i'} | i' \in I\}$.

By definition, $R_a \subset R$, $R_i \subset R$ and $R_a \cap R_i = \{r_{a,i}\}$.

If a peer knows R, it has access to a global knowledge on all ratings and who rated which items. For instance in order to compute similarity between items or between actors, one needs to use all ratings $r \in R$. This is the knowledge used by TrustWalker.

If a peer knows R_i, it has access to a global but anonymous knowledge on all ratings on a specific item, but not who made them. For instance in order to compute the ratings mean of an item i, one only needs to use anonymously all ratings $r_{a',i} \in R_i$ of this specific item, without knowing a'.

If a peer knows R_a, where a is the actor associated with this peer, it has access to a local knowledge on the ratings made by the actor a. For instance in order to compute the ratings mean of an actor a, the actor a only uses his/her own ratings $r_{a,i'} \in R_a$.

In section 4.3.1, the *actor's mean* requires to know R_a in eq.23 whereas the *item's mean* requires to know R_i in eq.24. TrustWalker requires to know R in order to compute items similarities.

The calculation of the ratings mean of an item does not respect our local assumption, however it respects our privacy objectives. It requires an anonymous global knowledge on ratings.

4.3.3 *Confidence on a default score*

Since default scores are not computed as regular scores, their confidence is computed specifically.

We compute default scores confidence based on the size and the variance of this ratings set, similarly as the size confidence and variance confidence defined in section 4.2.1, conditions 3 and 4 (page 48).

Definition 31: Default score ratings. *Let* $R_{default} = \{r_1, r_2, \dots, r_n\}$ *be set of ratings used to compute the default score:*

- $R_{default} = R_a$ *with the actor's mean strategy (eq.23) or*
- $R_{default} = R_i$ *with the item's mean strategy (eq.24).*

Depending on the default score strategy, $R_{default}$ *are the ratings from the actor* a *or on the item* i*.*

4.3.3.1 *Size confidence on a default score*

We use the *sigmoid* function defined in eq.15 in the same way as it is used in section 4.2.1.3:

$$c^s_{default} = sigmoid(|R_{default}| - 1) \tag{25}$$

4.3.3.2 *Variance confidence on a default score*

We compute the variance on $R_{default}$ similarly as section 4.2.1.4, but without weight:

$$c^{\sigma}_{default} = 1 - \frac{\sigma^2_{default}}{\sigma^2_{max}} \tag{26}$$

$$\sigma^2_{default} = \frac{\sum_{r \in R_{default}} (r - \mu)^2}{|R_{default}|} \tag{27}$$

With μ the mean of ratings $R_{default}$ and σ_{max} the maximum possible variance, *c.f.* section 4.2.1.4 Variance confidence (c^{σ}).

4.3.3.3 *Final confidence on a default score*

The final confidence coefficient on a default score is then the aggregation of the two coefficients:

$$c^{default}_{a,i} = c^s_{default} \times c^{\sigma}_{default} \tag{28}$$

This coefficient remains low when few ratings are used or when those ratings are heterogeneous.

4.4 COTCODEPTH SOCIAL SCORING

The previous coefficients are combined to bring our final scorer for social recommendation. Our scorer is the aggregation of all previous definitions: the Correlative and Trust with Confident k–Depth Social Scorer, CoTCoDepth Scorer for short (or $s^*_k(a, i)$ in eq.29).

4.4.1 *Definition*

CoTCoDepth is defined as $s^*_k(a, i)$ in the following equation:

$$s^*_k(a,i) = \begin{cases} r_{a,i} & \text{if } \exists r_{a,i} \\ \dfrac{\sum_{f \in \mathcal{F}^{k-1}_{a,i,\omega}} \omega_{a,f} \times s^*_{k-1}(f,i)}{\sum_{f \in \mathcal{F}^{k-1}_{a,i,\omega}} \omega_{a,f}} & \text{if } \nexists r_{a,i} \wedge \mathcal{F}^{k-1}_{a,i,\omega} \neq \emptyset \\ \mathit{default}(a,i) & \text{otherwise} \end{cases} \tag{29}$$

For a given item i, the weight ω for score propagation is defined in eq.30 and the weight $\omega^{(c)}$ for the original requester is defined in eq.31.

$$\omega_{a,f} = t_{a,f} \times \rho_{a,f} \tag{30}$$

$$\omega^{(c)}_{a,f} = t_{a,f} \times \rho_{a,f} \times c_{f,i} \tag{31}$$

With the confidence c defined as follow:

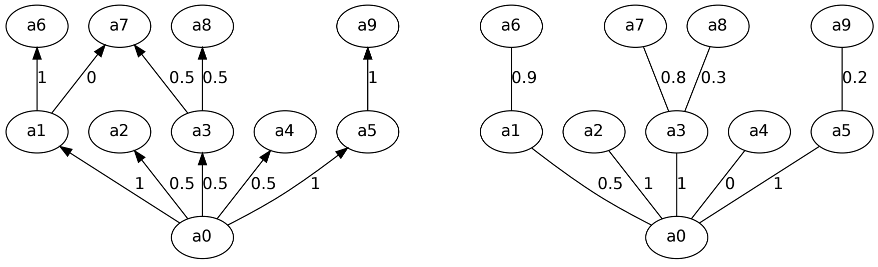
$$c_{a,i} = \begin{cases} c^{t}_{a,i} & \text{if } \exists r_{a,i} \\ c^{\mathit{size}}_{a,i} * \dfrac{c^{\mathcal{F}}_{a,i} + c^{\omega}_{a,i} + c^{\sigma}_{a,i}}{3} & \text{if } \nexists r_{a,i} \wedge \mathcal{F}^{k-1}_{a,i,\omega} \neq \emptyset \\ c^{\mathit{default}}_{a,i} & \text{otherwise} \end{cases} \tag{32}$$

4.4.2 *Example*

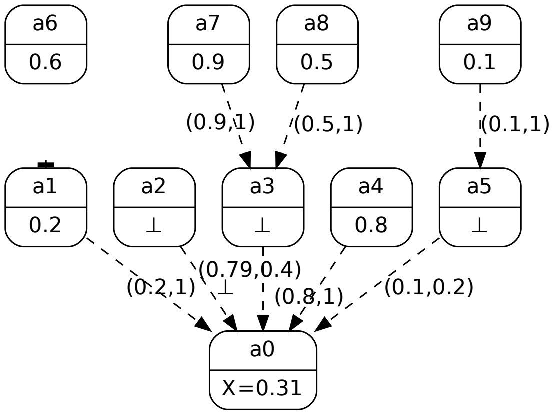
Figure 14 runs our motivating example with CoTCoDepth, using trust and similarity as well as confidence. For clarity purposes, we only use friends confidence, defined in section 4.2.1.1 and weight confidence, defined in section 4.2.1.2. Moreover we do not consider default scoring.

Results are similar to the ones in section 4.2.3, figure 13, except for a_3's influence. Almost all other trust values equal 1, a neutral value. Therefore we only detail computations for a_3 and then a_0.

a_3 computes a score using a_7 and a_8's scores. Both a_7 and a_8 are fully confident on their own scores, so a_3's friends confidence


a6
a7
a8
a9
1
0
0.5
0.5
1
a1
a2
a3
a4
a5
1
0.5
0.5
0.5
1
a0
a6
a7
a8
a9
0.9
0.8
0.3
0.2
a1
a2
a3
a4
a5
0.5
1
1
0
1
a0

(a) Trust network (b) Similarity network


a6
0.6
a7
0.9
a8
0.5
a9
0.1
(0.9,1)
(0.5,1)
(0.1,1)
a1
0.2
a2
⊥
a3
⊥
a4
0.8
a5
⊥
(0.79,0.4)
(0.2,1)
⊥
(0.8,1)
(0.1,0.2)
a0
X=0.31

(c) Score propagation

Figure 14: CoTCoDepth Example

is 1. a_3's weight confidence, being the maximum friends weight, is $0.8 \times 0.5 = 0.4$. a_3's score confidence is therefore:

$$c_{a_3,i_0} = c^{\mathcal{F}}_{a_3,i_0} \times c^{\omega}_{a_3,i_0} = 1 \times 0.4 = 0.4$$

Scores are propagated with their confidences to a_0, who takes them into account on the final score computation. Each weight contains trust and similarity:

$$\begin{aligned} s^*_2(a_0,i_0) &= \frac{\sum_{f \in \mathcal{F}^1_{a_0,i_0,\omega}} \omega_{a_0,f} \times c_{f,i_0} \times s^*_1(f,i_0)}{\sum_{f \in \mathcal{F}^1_{a_0,i_0,\omega}} \omega_{a_0,f} \times c_{f,i_0}} \\ &= \frac{(0.5 \times 1 \times 0.2) + (0.5 \times 0.4 \times 0.79) + (0 \times 1 \times 0.8) + (1 \times 0.2 \times 0.1)}{(0.5 \times 1) + (0.5 \times 0.4) + (0 \times 1) + (1 \times 0.2)} \\ &= 0.31 \end{aligned}$$

a_0 finally computes his/her final confidence on the score using friends and weight confidence:

$$\begin{aligned} c^{\mathcal{F}}_{a_0,i_0} &= \frac{(0.5 \times 1) + (0.5 \times 0.4) + (0 \times 1) + (1 \times 0.2)}{0.5 + 0.5 + 0 + 1} = 0.45 \\ c^{\omega}_{a_0,i_0} &= \max(0.5, 0.5, 0, 1) = 1 \\ c_{a_0,i_0} &= c^{\mathcal{F}}_{a_0,i_0} \times c^{\omega}_{a_0,i_0} = 0.45 \times 1 = 0.45 \end{aligned}$$

CoTCoDepth predicts to a_0 regarding the item i_0 a rating of 0.31 with a confidence of 0.45.

4.5 CONCLUSION

Our scorer uses a social network to recommend items. It is based on **local knowledge** such as the immediate social relations and local trust. We have then introduced firstly a **local similarity** based on friendship relations in order to promote friends with the same tastes and secondly a **confidence on the scoring** itself. In order to prevent the usual sparsity problem in trust-based recommendation, we propose a **default score** returned by actors. Our scorer **propagates scores and confidence** locally in the social network **without creating new relations**.

In the next chapter, we evaluate our system and compare it with existing trust-based or collaborative filtering recommender systems. We show that our local approach provides recommendations that are as accurate as the ones given by existing systems and provides more accurate recommendation for cold start users.

5

EVALUATION

Numbers are like people; torture them enough and they'll tell you anything.
— *Anonymous*

Contents

5.1 Campaigns 65
  5.1.1 Training set 65
  5.1.2 Leave one out 66
5.2 Dataset 67
  5.2.1 Epinions 67
  5.2.2 Flixster 68
  5.2.3 Appolicious 69
5.3 Implementation 70
  5.3.1 CoTCoDepth Scorer 70
  5.3.2 Evaluation 71
  5.3.3 Views 71
  5.3.4 Metrics 72
5.4 Influence of k and connectivity degree 72
  5.4.1 Epinions: Alchemy dataset 72
  5.4.2 Appolicious 75
  5.4.3 Flixster 76
  5.4.4 Conclusion 77
5.5 Comparison with existing approaches 78
  5.5.1 Scorers characteristics 78
  5.5.2 All actors 79
  5.5.3 Cold start users 80
5.6 Conclusion 80

Figures

Figure 15 Distribution on Alchemy 68
  (a) Ratings 68
  (b) Users' ratings 68
  (c) Items' ratings 68
Figure 16 Distribution on Flixster 69
  (a) Ratings 69
  (b) Users' ratings 69
  (c) Items' ratings 69
Figure 17 Distribution on Appolicious 70
  (a) Ratings 70
  (b) Users' ratings 70

(c) Items' ratings 70
Figure 18 Influence of training set size on coverage using Alchemy 73
Figure 19 Influence of connectivity degree on coverage using Alchemy 74
Figure 20 Influence of connectivity degree on precision using Alchemy 74
Figure 21 Influence of connectivity degree on coverage using Appolicious 75
Figure 22 Influence of connectivity degree on precision using Appolicious 76
Figure 23 Influence of connectivity degree on coverage using Flixster 77
Figure 24 Influence of connectivity degree on precision using Flixster 77

Tables

Table 2 Results for all actors on Epinions 79
Table 3 Results for cold start users on Epinions . 80

In the previous chapter, we have defined our local social-based recommender system. This chapter evaluates our approach regarding inner parameters, such as the depth propagation or the default scoring. It also compares our approach with existing ones.

Section 5.1 depicts our evaluation protocol through two mains campaigns: training set and leave-one-out. The training set campaign evaluates only a part of the ratings and allows to evaluate an algorithms in function of dataset sparsity. The leave-one-out campaign evaluates the whole dataset and is reproducible.

Section 5.2 describes the datasets used in our evaluation. Three different Epinions' datasets: Alchemy [RD02], Trustlet [MA06] and RED, *c.f.* section B.1. We have also used a Flixster dataset [JE10] and a Appolicious dataset, *c.f.* section B.2.

Section 5.3 gives a brief description of the scorers implementations we have made in order to run the evaluation.

Finally, we present and discuss our results in sections 5.4 and 5.5.

In section 5.4, we have run our scorers on three datasets (Alchemy, Flixster and Appolicious) in order to compare the influence of depth propagation, actors' connectivity and dataset density on coverage and precision. We have used the training set campaign on the Alchemy dataset in order to measure the impact of the training set size on coverage.

Section 5.5 provides a comparison between our approach and the state of the art using the Alchemy dataset. We compare our scorers with different configurations: pure trust-based without default score, pure local with actors mean ratings default score and pure anonymous with items mean ratings default score.

5.1 CAMPAIGNS

In order to evaluate our algorithms and compare them with the state of the art, we have run two different kinds of evaluation campaigns: the "training set" and the "leave one out".

5.1.1 *Training set*

The training set evaluation campaign is a classical method. The dataset is split into two datasets: the evaluation set and the training set. The training set is used by the algorithms to predict the scores of the evaluation set.

The difference between predicted scores and real ratings contained in the evaluation dataset is used as an indicator of the quality of the prediction algorithm using the evaluation metrics, described in section 5.3.4. The training set is also called the context.

The split is based on ratings, not on actors nor items. The two sets are disjoint.

This campaign is simple to implement since the training set is clearly defined and does not change during the whole evaluation process. Moreover, the evaluation set is smaller than the whole dataset and therefore the evaluation is faster. However the split is randomly made, therefore this campaign must be $k-$cross validated with k random different splits and a mean computed on the evaluation metrics.

To process this campaign, we shuffle all the ratings of the different datasets, then we split the ratings in two groups. We consider one experience per shuffle. A $k-$cross validation implies k experiences. Then for each experience, we randomly split the dataset into two parts with different sizes: 20 %-80 %, 50 %-50 %, 80 %-20 % and 90 %-10 %. The first group becomes the training set: we use these ratings as a context known by the scorers to predict the ratings contained in the evaluation set. The second group becomes the evaluation set: we try to predict these ratings.

We then run this experience for each ratio. The smaller the training set, the harder the predictions. The size of the training dataset gives an indications on the robustness of the recommender system against sparsity.

5.1.2 *Leave one out*

In this campaign, we use the whole dataset. For each rating, we consider that rating as the evaluation set and the other ratings as the training set. Once that rating is predicted, we do the same with the next rating. That is, the training set contains all ratings, then for each rating, we remove it from the training set, try to predict it and finally put it back in the training set.

Once this is done for all ratings in the dataset, results are aggregated and evaluated with the metrics defined in section 5.3.4, page 72.

There is no random selection so this campaign is reproducible. No need for $k-$cross validation. However it is harder to implement since the training set is not constant during the whole evaluation.

Since similarity is computed using ratings, it must be recomputed for each removed rating in the dataset. Therefore, once a rating is removed, similarity coefficients associated with the actor and/or the item are recomputed.

We denote the "leave one out" campaign as the "100 %" training set, where the training set contains all but one rating. Our evaluation is therefore based on five training sets with respectively 20, 50, 80, 90 and 100 % of training ratings.

5.2 DATASET

5.2.1 *Epinions*

Epinions [1] is a website providing "unbiased reviews by real people". It gathers reviews from users on items. A review contains a rating, a description of the review and some other pieces of information. Items are various kinds of items, from books to toys. They belong to categories and can be bought directly from the website, through well-known sellers (Amazon, Target, etc.). Users follow other users in order to get recommendations from them. It forms a trust network explicitly built by users.

To the best of our knowledge, Epinions datasets are the first datasets that have been published containing both a trust network and ratings on items. Therefore Epinions is well studied in trust-based recommender systems and is often used in order to compare existing approaches together, which is the case in section 5.5, page 78.

In this thesis, we have used three different extractions of the Epinions website:

- *Alchemy*: this dataset has been published in 2002 by [RD02],
- *Trustlet*: this dataset has been published in 2006 by [MA06],
- *RED*: as Rich Epinions Dataset, this dataset has been extracted by ourselves in 2011 [MGML11].

The two first datasets contain only ratings from users on items and trust values between users. The last dataset contains more data on Epinions, such as item categories, item descriptions, user expertises, etc., *c.f.* appendix B.1, page 120.

Since Alchemy is the most used dataset in trust-based recommender systems evaluations, we use this one in this chapter in order to evaluate our approach and compare it with existing systems.

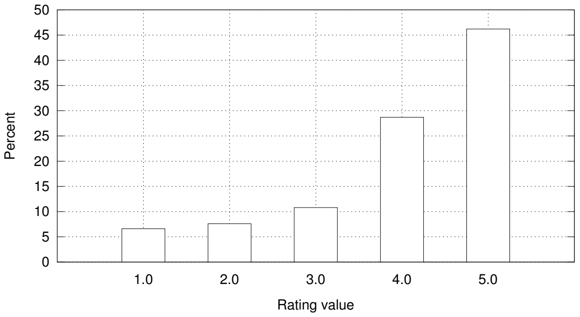
This dataset contains 104k items including 54k items with only one rating, which represents 10 % of all ratings. Classical collaborative filtering approaches, including trust-based ones, cannot predict any score for those items since no other actor has rated the item and once the rating has been removed, with the leave one out campaign, no similarity is computable. Those approaches can reach 90 % coverage at maximum.

Figure 15 shows the ratings distribution of the Alchemy dataset. The distribution is shown per rating (figure 15a), per users' ratings count (figure 15b), and per items' ratings count (figure 15c). The two other datasets have significantly the same distribution.

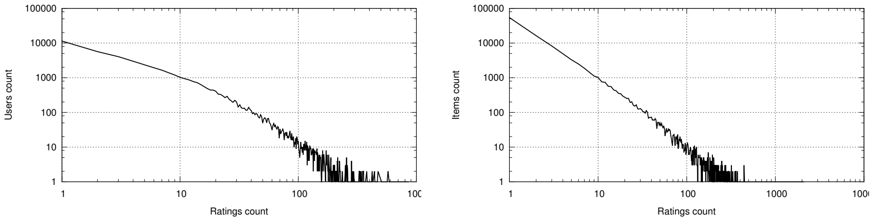
Alchemy and Trustlet are referenced in the literature. Nevertheless they have some drawbacks. Firstly, they are really sparse, both

---

[1] `http://www.epinions.com`



(a) Ratings



(b) Users' ratings

(c) Items' ratings

Figure 15: Distribution on Alchemy

in terms of ratings and trust values: between 10 and 15 ratings per user and about 10 trust values per user in average, depending on the dataset. Secondly, they only contain trust values, explicitly set by users on other users that are likely to provide useful recommendations. Trust values are directed and are not based on friendship relations.

5.2.2 *Flixster*

Flixster[2] is a website focusing on "discovering, watching and collecting movies". It is a movie platform offering video on demand.

Users can search for movies in order to buy or build a collection of movies. Movies receive ratings from users and reviews from critics. Finally, recommended movies appear in the front page when the user logs in the website. Since users can sign up with their Facebook account, Flixster uses friends ratings in recommendations.

To the best of our knowledge, Flixster is the only dataset providing ratings and friendship relations coming from Facebook. The dataset released by [JE10] provides ratings and friendship relations. Ratings are associated with timestamps. However no information is provided on users nor movies.

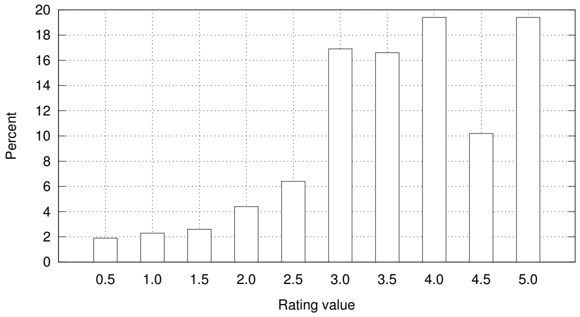
The dataset contains 147 612 users, 48 794 movies, 8 196 077 ratings, thus a 0.11 % density, and 5 897 094 friendship relations. It is our densest dataset.

---

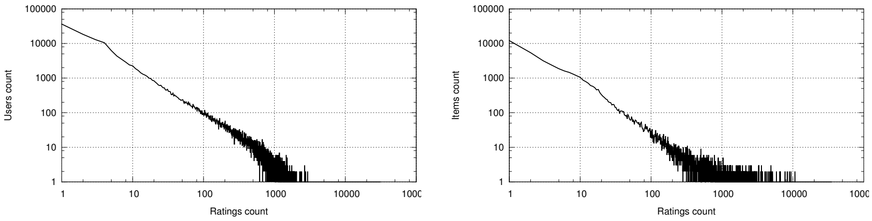
[2] `http://www.flixster.com`

A user has rated in average 55.5 movies and has 40 friends. However, no matter how dense this dataset is, the similarity is only computable on average for 1.5 friends of a user.

Figure 16 shows the ratings distribution on the Flixster dataset. Ratings go from 0.5 to 5 by 0.5. The distribution is shown per rating (figure 16a), per users' ratings count (figure 16b), and per items' ratings count (figure 16c).



(a) Ratings



(b) Users' ratings

(c) Items' ratings

Figure 16: Distribution on Flixster

5.2.3 *Appolicious*

Appolicious[3] is a mobile applications catalog/seller, for iPhone, iPad or Android mobile phones. It contains reviews made by users and recommends applications based on the users profile.

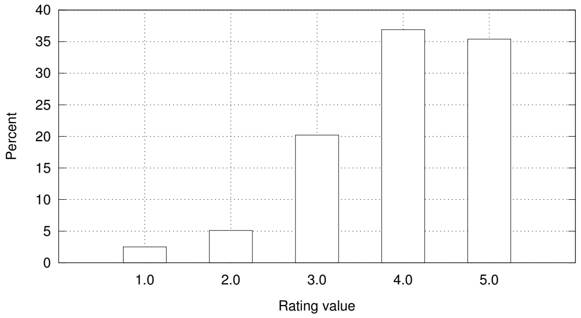
Users browse the catalog and build a list of owned applications and give reviews or ratings on them. They can follow other users of the community. Anyone can follow or block another user. The website provide a Facebook connection in order to import automatically Facebook friends in the community and follow them. Unfortunately the provided dataset does not distinguish Facebook friends from followed users.

This dataset contains more information than the previous ones. It is also smaller, with 4 058 users, 8 935 applications, 28 963 ratings and 20 815 following links. The density of this dataset is 0.08 %. For more information on that dataset, refer to appendix B.2, page 126.

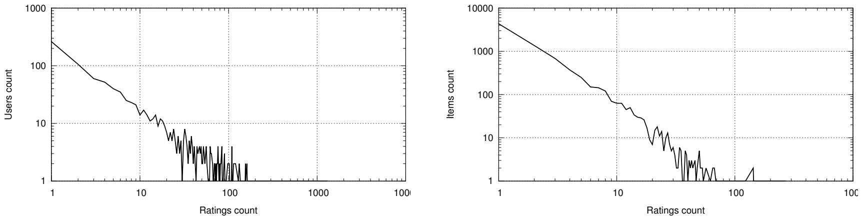
---

[3] `http://www.appolicious.com`

Figure 17 shows the ratings distribution on the Appolicious dataset. The distribution is shown per rating (figure 17a), per users' ratings count (figure 17b), and per items' ratings count (figure 17c).



(a) Ratings



(b) Users' ratings

(c) Items' ratings

Figure 17: Distribution on Appolicious

5.3 IMPLEMENTATION

This section focuses on the implementation details of the evaluation. We describe in section 5.3.1 the CoTCoDepth versions that we have implemented and evaluated. Section 5.3.2 specifies which datasets were used in which evaluation and which approaches from the state of the art were implemented and compared. In order to observe prediction effectiveness on specific kinds of actors, section 5.3.3 defines views that categorize actors depending on their ratings and connectivity. Finally, the metrics we have used in our evaluation are described in section 5.3.4.

5.3.1 *CoTCoDepth Scorer*

In order to evaluate the social approach, we have implemented different versions of our CoTCoDepth Scorer, as defined in chapter 4.

We have propagated scores up to $k \in \{1, 2, 3\}$, without default score ($P_{default} = 0$) and with actor mean or item mean as default score ($P_{default} = 0.02$):

- CoTCoD1 propagates to $k = 1$ without default score
- CoTCoD2 propagates to $k = 2$ without default score

- CoTCoD3 propagates to $k = 3$ without default score
- CoTCoD3_a propagates to $k = 3$ using actor mean rating as default score, with $P_{default} = 0.02$
- CoTCoD3_i propagates to $k = 3$ using item mean rating as default score, with $P_{default} = 0.02$
- CoTCoD3_{ia} propagates to $k = 3$ using item mean rating as default score or actor mean rating if the item mean is not computable, with $P_{default} = 0.02$

5.3.2 *Evaluation*

We have then observed the influence of k and of the number of friends (*i. e.* connectivity degree) on coverage and precision. We have run our scorers on the three datasets – Alchemy, Flixster, Appolicious – in order to see how dataset density and ratings distribution may influence the coverage and the precision. As one can expect, the denser, the less deep we need to propagate in the social network in order to have a good coverage.

We have also compared our scorers with the approaches described in sections 2.2.1 and 2.3 (pages 17 and 18): UserBasedCF, ItemBasedCF, MoleTrust, RandomWalk, and TrustWalker. In order to provide a fair comparison, we have used $max - depth = 3$ for MoleTrust, RandomWalk and TrustWalker. We have used the Alchemy dataset of Epinions for this comparison, since it is the most used in trust-based recommender systems evaluation. Moreover, Flixster is so dense that running all those algorithms requires too much computing resources to provide results and Appolicious does not contain enough users to provide statistically reliable results compared to the other datasets.

5.3.3 *Views*

The Alchemy dataset is presented in section 5.2. We have split the dataset into views to observe the influence of the connectivity degree in the social network, *c.f.* appendix B.1.3.4 (page 124):

- actors with at least one friend but with less than five friends are “weakly connected” (47 % of actors with 22 % of ratings)
- actors with five to nine friends are “fairly connected” (11 % of actors with 12 % of ratings)
- actors with ten friends or more are “highly connected” (18 % of actors with 57 % of ratings)

23 % of actors with only 8 % of ratings do not have any relation.

5.3.4 *Metrics*

In this chapter, we provide four statistical metrics: Coverage, Root Mean Square Error (RMSE), Precision (based on RMSE) and F-Measure.

Let r_n be the n^{th} rating and N the total number of ratings. Let p_n be the predicted rating for the n^{th} rating and N' the total number of predicted ratings.

The coverage is the proportion of predicted ratings regarding all ratings to predict. It does not indicate the quality of predictions but shows how many predictions an algorithm can fulfill.

$$\mathit{Coverage} = \frac{N'}{N} \times 100 \tag{33}$$

The RMSE represents the average error of the prediction. It is basically the error standard deviation without mean. The lower the RMSE, the more accurate the prediction. However it is only computable with predicted scores.

$$\mathit{RMSE} = \sqrt{\frac{\sum_{n=1}^{N} (p_n - r_n)^2}{N}} \tag{34}$$

The precision is the counterpart of the RMSE inversing the result: the higher the precision, the more accurate the prediction. Note that *range* is the rating interval, *i. e.* the largest possible error. In Epinions, $\mathit{range} = 4$:

$$\mathit{Precision} = 1 - \frac{\mathit{RMSE}}{\mathit{range}} \tag{35}$$

The *F-Measure* F_1 combines coverage and precision [JE09]:

$$F_1 = \frac{2 \times \mathit{Precision} \times \mathit{Coverage}}{\mathit{Precision} + \mathit{Coverage}} \tag{36}$$

5.4 INFLUENCE OF k AND CONNECTIVITY DEGREE

We have run our scorers described in section 5.3.1 with the three datasets listed in section 5.2: Alchemy (Epinions), Appolicious and Flixster. We have run those evaluations using the five training sets with respectively 20, 50, 80, 90 and 100 % of training ratings, in order to see the impact of the dataset sparsity on the results.

5.4.1 *Epinions: Alchemy dataset*

We have run more evaluations with the Alchemy dataset and provide results with more details in this section.

5.4.1.1 *Coverage*

Figure 18 shows coverage depending on the training set size and the depth propagation. TrustAll is an algorithm taking into account all actors ratings, as if the actor trusted every one else (*c.f.* section 2.3.2): it indicates the highest coverage possible with pure trust-based approaches. This figure only takes into account actors with at least one trust relation.

Increasing the training set size improves coverage, since there are more ratings to rely on. Depth propagation is also an important factor, there is an almost 40 % coverage gap between $k = 1$ and $k = 2$. Obviously, immediate vicinity is not sufficient to predict scores. Propagating scores in the social network offers an efficient solution to improve coverage.

More important, with a depth propagation of 3, the coverage is almost as high as with the TrustAll approach: 83.56 % of ratings made by actors with at least one trust relation are covered by CoTCoD3, whereas TrustAll covers 90.57 %. There are only 7 points of difference despite the latter uses all actors in order to predict ratings.

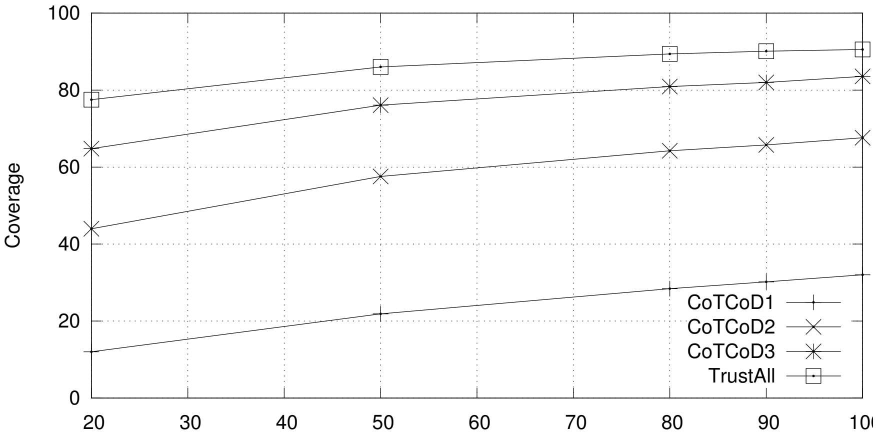


Figure 18: Influence of training set size on coverage using Alchemy

Figure 19 shows the details of the coverage depending on the actors connectivity, with a 100 % training set. Increasing either k or the number of friends improves coverage, as one would expect. The more links in the social network, the higher the coverage.

The *default score* enhances the coverage as expected. Returning an actor mean rating is even more covering than returning an item mean rating. This is due to items with only one rating: since the only available rating for the item is removed, no mean is computable. But most actors still have other ratings to use to compute their mean, enhancing the coverage. With this approach, nearly 100 % ratings are predicted for fairly and highly connected actors.

Using a sparse trust network implies that we need to propagate further in the graph in order to cover enough ratings. We can see

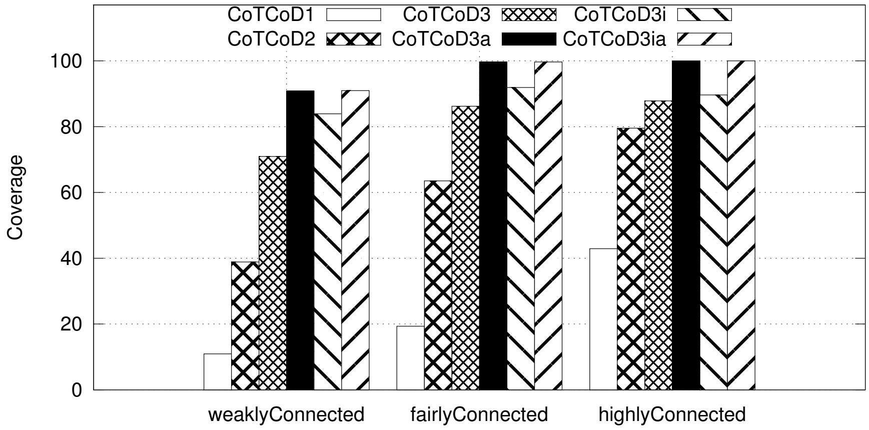


Figure 19: Influence of connectivity degree on coverage using Alchemy

that increasing k improves coverage, since it increases the number of actors involved in the recommendation. To avoid the small world effect [Mil67], we did not propagate scores further than $k = 3$. However, propagating to $k = 3$ with $P_{default} = 0$ covers more than 86 % ratings made by actors trusting more than four actors, despite using a sparse trust network. Moreover, introducing a low probability to return a default score ($P_{default} = 0.02$) increases the coverage to reach more than 90 % ratings, all actors included, and almost 100 % ratings from actors trusting more than four actors.

5.4.1.2 *Precision*

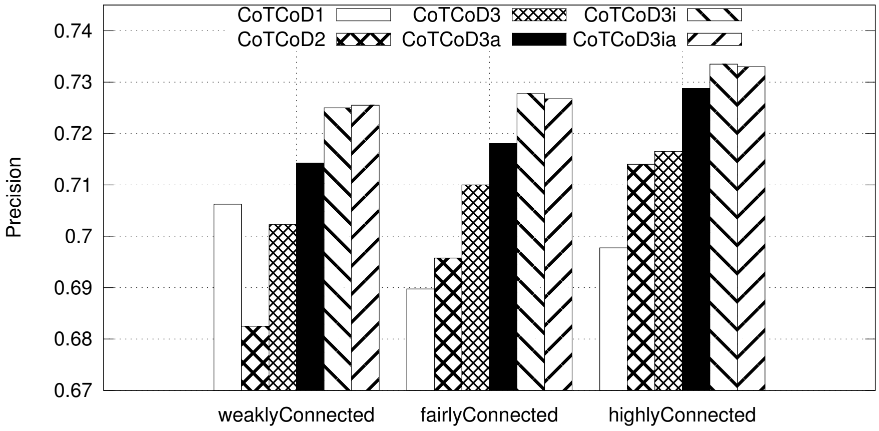


Figure 20: Influence of connectivity degree on precision using Alchemy

Figure 20 shows that increasing the number of friends improves the precision (*c.f.* eq.35 page 72), the more an actor is connected the more accurate the recommendations, as one would expect. Being highly connected implies having more recommendations, therefore the aggregation returns a more reliable answer. The CoTCoD1 precision is not significant since the coverage is too low, specially for weakly connected users.

However propagating further in the network also improves the precision, which was less expected. Immediate friends should be more likely to provide relevant recommendations. Here again, a deeper propagation leads to more scores, which explains a better precision.

We saw in the previous section that item mean rating as default score provides a lower coverage than actor mean rating. However precision is higher, which is legitimate since it uses ratings on the requested item. Therefore using anonymous aggregated data enhances significantly the precision of our approach. Moreover, returning items mean ratings if available or actors mean ratings otherwise (CoTCoD3_{ia}) gets the coverage of CoTCoD_a with a precision as high as CoTCoD_i: it is thus the best approach.

5.4.2 *Appolicious*

5.4.2.1 *Coverage*

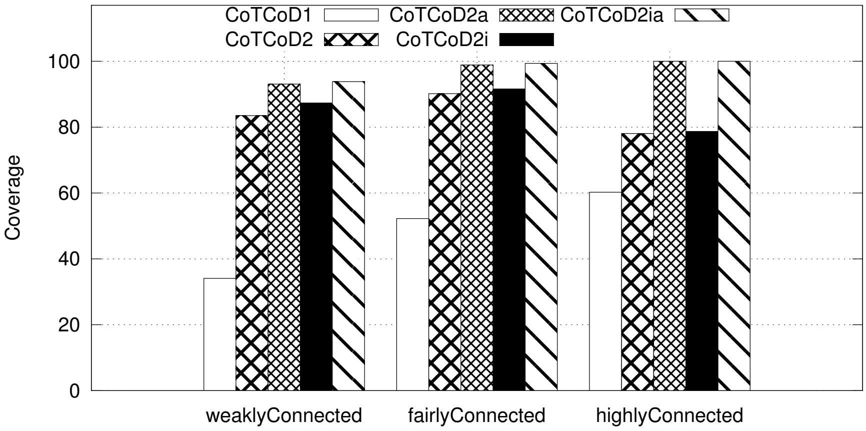


Figure 21: Influence of connectivity degree on coverage using Appolicious

Figure 21 shows that asking up to friends of friends ($k = 2$) is enough to reach a high coverage. Since this dataset is denser than the Epinions' one, we do not need to propagate far in the social network in order to predict scores.

The *default score* still enhances the coverage, but not as much as previously, since our approach without default scoring covers enough ratings. With actor mean rating default scoring, nearly 100 % ratings are predicted for fairly and highly connected actors.

Surprisingly, highly connected actors have with $k = 2$ a lower coverage than other actors, whereas the coverage increases with $k = 1$. The missing delta may have disappeared because those actors are the ones giving the most ratings, therefore they have rated many items that have only one rating. Therefore when we remove that rating for the evaluation, no one can provide any score for that item. This is supported by the fact that CoTCoD2 and CoTCoD2$_{\mathrm{i}}$ have the same coverage: the missing scores come from items with only one rating, the one for which the mean is no longer computable.

With a dense enough dataset, propagating only to friends of friends is enough to provide recommendation. The default score is here still efficient, but not as useful as with Epinions.

5.4.2.2 *Precision*

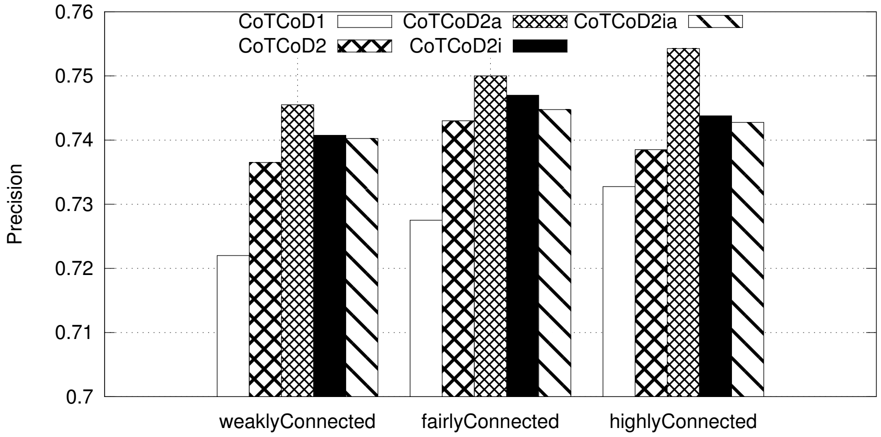


Figure 22: Influence of connectivity degree on precision using Appolicious

Figure 22 shows that

- Increasing the number of friends improves precision (*c.f.* eq.35), except for highly connected actors, as explained above.
- Propagating further in the network also improves precision.
- Unlike with the Epinions dataset, the default scoring using items mean ratings does not improve precision that much.
- However, the actors mean ratings default scoring improves both coverage and precision.

5.4.3 *Flixster*

5.4.3.1 *Coverage*

Figure 23 shows that with Flixster asking up to friends of friends ($k = 2$) is enough to reach almost all actors. This dataset is the densest one and actors have in average 40 friends. This explains why a shallow propagation reaches a high coverage.

As with the Appolicious dataset, propagating only to friends of friends with a dense dataset is enough to provide recommendation.

The *default score* does almost not impact the coverage. Without default score, nearly 100 % ratings are predicted for fairly and highly connected actors.

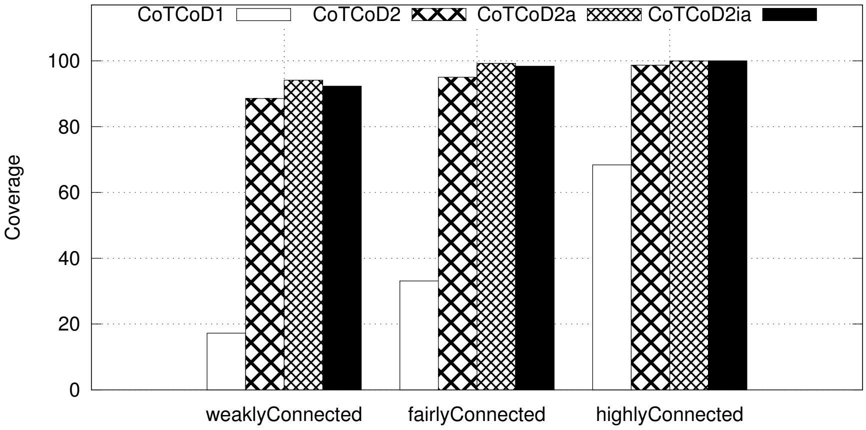


Figure 23: Influence of connectivity degree on coverage using Flixster

5.4.3.2 *Precision*

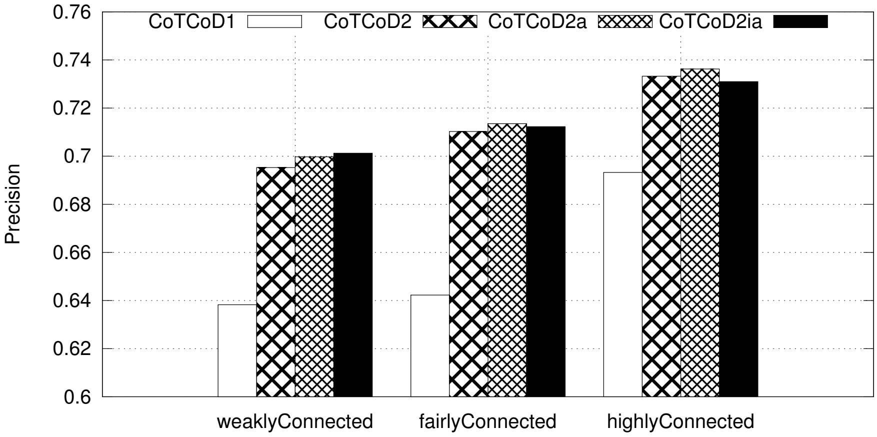


Figure 24: Influence of connectivity degree on precision using Flixster

In accordance with figure 24, we observe similar remarks with Flixster and Appolicious datasets, *c.f.* section 5.4.2.2.

The main difference is regarding default scores. Both items mean ratings and actors mean ratings do not really improve neither precision nor coverage. Since this dataset is the densest one, this confirm that default scoring strategies are not adapted to dense datasets.

5.4.4 *Conclusion*

This section shows the influence of sparsity, connectivity and score propagation.

Regarding a sparse dataset such as Alchemy, we only need to propagate up to $k = 3$ in order to reach the suitable coverage and precision. However this is not true for all actors: we provide good recommendation for highly connected actors with $k = 2$. Moreover, default scoring increases both coverage and precision. It therefore reduces the required propagation depth.

Regarding dense datasets, such as Appolicious and Flixster, a depth propagation of $k = 2$ is sufficient for all actors. Obviously the use of dense social networks instead of sparse trust networks enhances coverage. Default scoring is not needed in that case.

Our approach is compatible with both trust and social networks. Default scoring copes with trust networks sparsity whereas low depth propagation is enough for dense social networks.

In the following section, we compare our approach with the state of the art, using the commonly used Epinions Alchemy dataset.

5.5 COMPARISON WITH EXISTING APPROACHES

In order to compare our approach with others, we have implemented UserBasedCF, ItemBasedCF, MoleTrust3, RandomWalk3 and TrustWalker3, using a trust propagation of 3 for the latter. We want to minimize the trust propagation since we target decentralized architectures and this propagation is costly, 3 is the best trade-off between cost, accuracy and coverage.

Since Epinions is the most used dataset for trust-based recommendation evaluation, we have used the Alchemy dataset with all approaches. Section 5.5.1 describes the characteristics of the evaluated scorers. Results on Epinions dataset are described in section 5.5.2 for all actors and in section 5.5.3 for cold start users.

5.5.1 *Scorers characteristics*

We have implemented our approach both without (CoTCoD3) and with (CoTCoD3_a and CoTCoD3_{ia}) default scoring in order to provide honest comparison:

- UserBasedCF and ItemBasedCF are classical collaborative filtering recommender systems, *c.f.* section 2.2.1 page 17.
- CoTCoD3, MoleTrust3 and RandomWalk3 are pure trust-based approaches using only trust values in the Epinions trust network. Only CoTCoD3 and RandomWalk3 are purely local.
- CoTCoD3_a is a pure local approach, fully P2P compliant but using a default scoring strategy, unlike CoTCoD3.
- CoTCoD$_{ia}$ is a pure anonymous approach, using item mean rating as default score, it need some anonymous knowledge on items ratings.
- TrustWalker3 uses item-based similarity, thus comparable with our default score where an actor still returns a rating when he/she did not rate the considered item.

5.5.2 *All actors*

Method	**RMSE**	**Cov.**	**F_1**	**Knowledge**
UserBasedCF	1.138	69.09	0.703	global
ItemBasedCF	1.364	65.64	0.658	global
MoleTrust3	1.100	77.25	0.748	extended-local
RandomWalk3	1.271	53.44	0.599	local
CoTCoD3	1.150	77.25	0.741	local
$\mathrm{CoTCoD3}_a$	1.105	90.50	0.804	local
$\mathrm{CoTCoD3}_{ia}$	1.078	90.56	0.809	anonymous
TrustWalker3	1.092	85.99	0.788	global

Table 2: Results for all actors on Epinions

Table 2 indicates the RMSE, the coverage and the f-measure of all actors ratings prediction.

Regarding only pure local approaches, $\mathrm{CoTCoD3}_a$ provides the best results both in terms of precision and coverage. RandomWalk3 is not effective enough with a low propagation depth of 3. That kind of approach usually propagates up to depth 6. MoleTrust3 offers the same coverage than CoTCoD3 with a lower error but it relies on extended-local knowledge. Moreover $\mathrm{CoTCoD3}_a$ still has a higher f-measure than MoleTrust3. $\mathrm{CoTCoD3}_a$ helps improving coverage but also accuracy. As stated by [MA07a], the Epinions dataset contains mostly 5 as rating value which explains why returning an average rating improves accuracy.

Regarding classical collaborative filtering approaches, both UserBasedCF and ItemBasedCF are outperformed by the trust-based approaches, since similarity is seldom computable with sparse datasets such as Epinions [JE09, MA07a].

Based on local and global knowledge, TrustWalker has the second lowest error with 1.092. However it covers less ratings than $\mathrm{CoTCoD3}_a$, based on local knowledge, and $\mathrm{CoTCoD3}_{ia}$, based on local or anonymous knowledge[4].

$\mathrm{CoTCoD3}_{ia}$ is the best system in our evaluation with the lowest error and the highest coverage, therefore the highest f-measure. The default score strategy alleviates coverage limitation and is compatible with our architecture and privacy assumptions. It provides scores to items that cannot be recommended with classical collaborative filtering and trust-based approaches.

---

[4] The accuracy of our TrustWalker implementation is coherent with [JE09], while coverage is 10 % lower. We do not understand the coverage indicated in [JE09] since, as stated in section 5.2.1, 90 % should be the maximum coverage with classical collaborative filtering approaches using a leave one out evaluation campaign and the Epinions dataset.

Our approach is thus adapted for sparse networks such as Epinions, where actors need to explicitly express that they trust other people in order to connect with them, without propagating deeply in the network and only based on local or local and anonymous information.

5.5.3 *Cold start users*

Method	**RMSE**	**Cov.**	**F$_1$**	**Knowledge**
UserBasedCF	1.248	15.13	0.248	global
ItemBasedCF	1.639	21.49	0.315	global
MoleTrust3	1.167	50.51	0.590	extended-local
RandomWalk3	1.288	37.74	0.485	local
CoTCoD3	1.196	50.51	0.587	local
CoTCoD3$_{a}$	1.145	65.05	0.681	local
CoTCoD3$_{ia}$	1.103	65.48	0.688	anonymous
TrustWalker3	1.287	67.50	0.677	global

Table 3: Results for cold start users on Epinions

Table 3 compares those approaches regarding only cold start users. With those particular actors, who do not have enough ratings to perform well with classical collaborative filtering, CoTCoDepth outperforms all other local and global algorithms in terms of precision and coverage, except for TrustWalker3 which covers 2 % more ratings.

RandomWalk3 and the global approaches do not cover enough ratings to be significant. TrustWalker3 suffers a high error.

Here again CoTCoD3$_{ia}$ provides the best precision and coverage for cold start users. The pure local algorithm CoTCoD3$_{a}$ provides the second highest f-measure, just below CoTCoD3$_{ia}$.

5.6 CONCLUSION

In this chapter, we have evaluated different versions of our approach in order to measure the impacts of sparsity, connectivity and score propagation on the results. As expected, the deeper we propagate, the better the recommendation in terms of coverage and accuracy, since actors aggregate more recommendations. Our default scoring strategies also greatly improve coverage and accuracy with sparse datasets like Epinions. With denser datasets, like Appolicious and Flixster, this is not always the case.

The pure trust-based version of our approach, without default scoring, does not cover all ratings with the Epinions dataset. But we have shown that with an actual social network, *e.g.* Facebook with Flixster, a score propagation to $k = 2$ covers more than 97 % users. With default scoring, almost 99.5 % ratings are predicted.

We have also compared our approach with the state of the art. Clearly, classical collaborative filtering recommender systems do not perform well with sparse datasets, especially for cold start users. Trust-based recommender systems do better, even with a sparse trust network. Additional features such as the item-based similarity for TrustWalker or the default scoring for CoTCoDepth significantly improve the results.

Regarding all approaches in this evaluation, CoTCoD$_{ia}$ is the best regarding coverage and accuracy. This scorer requires however some anonymous knowledge on items ratings. When such knowledge is not available, CoTCoD3$_a$ is the best alternative. It is purely local and provides one of the best f-measure.

To sum up, regarding the kind of available data, we provide 3 versions of our CoTCoDepth scorer. The first one, CoTCoD3, is purely local and trust-based. It is the best in this category. The second one, CoTCoD3$_a$, is purely local but not only trust-based, since it adds default scoring. It is one of the best scorers, especially regarding only local ones. The third one, CoTCoD3$_{ia}$, respects our assumptions on privacy and provides the highest precision, coverage and f-measure in our evaluation.

This chapter proposes a centralized evaluation in order to compare our approach with the state of the art. Since CoTCoDepth is P2P compatible, we propose in the following chapter heuristics that refine our approach in particular regarding P2P architectures. Moreover, some heuristics aim at improving network usage and therefore we evaluate the impact of our approach on this criterion.

6

HEURISTICS

It is the excitement of becoming - always becoming, trying, probing, falling, resting, and trying again - but always trying and always gaining.
— *Lyndon B. Johnson*

Contents

6.1 Heuristics evaluation protocol 85
6.2 Extended similarity 86
  6.2.1 Definition 86
  6.2.2 Evaluation 87
6.3 Relative scoring 87
  6.3.1 Relative score propagation 88
  6.3.2 Relative scoring evaluation 88
6.4 A New Hop 89
  6.4.1 Score propagation with hops 90
  6.4.2 Hops evaluation 90
6.5 Friends selection 91
  6.5.1 Random friends selection 92
  6.5.2 Random raters selection 93
  6.5.3 Weight influence 96
  6.5.4 Conclusion 98
6.6 Expertise 98
  6.6.1 Friends expertise 98
  6.6.2 Global expertise 99
  6.6.3 Expertise evaluation 99
6.7 Conclusion 100

Figures

Figure 25 Influence of correlation on precision using Alchemy 87
Figure 26 Absolute vs. Relative scores on Alchemy 89
Figure 27 k−Depth Social Scoring with Hops; Example with $\alpha = 0.5$ 90
Figure 28 Influence of α on the RMSE, using hops . 91
  (a) $k = 2$ 91
  (b) $k = 3$ 91
Figure 29 Random friends selection evaluation using Alchemy 93
  (a) Coverage 93
  (b) RMSE 93
  (c) Execution time 93

Figure 30 Random friends selection evaluation using Flixster . . . 94
(a) Coverage . . . 94
(b) RMSE . . . 94
(c) Execution time . . . 94
Figure 31 Random raters selection evaluation using Alchemy . . . 95
(a) RMSE . . . 95
(b) Execution time . . . 95
Figure 32 Random raters selection evaluation using Flixster . . . 96
(a) RMSE . . . 96
(b) Execution time . . . 96
Figure 33 Weight influence compared to random raters selection . . . 97

Tables

Table 4 RMSE with relative or absolute score depending on k . . . 89
Table 5 Results for expertise heuristics on Epinions 100

Chapter 4 describes our social recommendation algorithms based on friends ratings. It introduces score propagation and some others algorithms implemented on this propagation. Chapter 5 evaluates our approach and compares it with existing systems.

In this chapter, we define heuristics that modify the score propagation in some ways, in order to improve precision or coverage or in order to reduce the network overload caused by propagation. Indeed, although it is local-based and requires few processing resources, the propagation in the social network can become quite heavy, especially for actors with a lot of friends.

This chapter is organized as follows: each section introduces and describes heuristics aiming at improving the system on one dimension. Then, each section contains an evaluation of those heuristics providing accuracy, coverage or resources metrics.

Section 6.1 describes briefly our evaluation protocol for this chapter. Section 6.4 proposes a new score propagation where all friends of friends get reached by the request, even if intermediate friends have ratings. Section 6.5 describes an heuristic where actors do not ask all their friends, but only a subset of them. Section 6.6 proposes to take into account expertise from actors to item categories, therefore not treating all friends in the same way, depending on what kind of item we are recommending.

6.1 HEURISTICS EVALUATION PROTOCOL

In order to compare the proposed heuristics, we provide an evaluation for each of them. Three metrics are used: coverage and RMSE, as described in section 5.3.4 page 72 and the average duration of a rating prediction.

The average duration of a rating prediction is simply the time for a specific algorithm to predict all ratings divided by the number of ratings to predict. Unpredicted ratings are also taken into account since it requires times in our approach to detect that the rating will not be predicted.

This metric is not used in the comparative evaluation, chapter 5, since some algorithms are more optimized and/or more adapted to the simulation architecture than others. Thus it is not fair to use time to compare them. However, for a given algorithm, here CoTCoDepth, it makes sense to compare heuristics applied on it based on the time to predict ratings.

The evaluation has been made using a 90 % training set and a 10 % evaluation set with the Alchemy dataset, described in section 5.2.1 (page 67). The training set campaign evaluation is explained in section 5.1.1 (page 65).

6.2 EXTENDED SIMILARITY

We have seen in section 4.1.3, page 46, that our approach is compatible with local similarity, *i. e.* similarity computed between friends. However, the intersection between similar actors and friends is usually small. In Alchemy, an actor has in average 11 friends and among them, a correlation coefficient is computable in average for only 2 of them. In Flixster, an actor has in average 48 friends with a correlation computable with only 3 of them.

6.2.1 *Definition*

Obviously the similarity approach is not adapted and is solely based on the default similarity coefficient that the system uses. However, in order to increase the number of friends for whom a similarity is computable, we have introduced the extended similarity.

Similarity is based on a correlation coefficient computed between common raters of items. In order to compute the similarity between two actors, we take the intersection between the items rated by the first one and the items rated by the second one. Then, we use the ratings on those items to compute the Pearson correlation coefficient. Since common raters and friends almost never overlap, this approach is not effective enough.

Extended similarity is computed between friends and does not consider common raters anymore. Then, the correlation coefficient is computed using all items rated by either one or the other actor. However, for missing ratings, the rating is predicted thank to our social approach. Basically when a rating is missing, we try to predict it using the friends of the actor, including the one we want to compute similarity with.

Therefore the extended Pearson correlation coefficient ρ' becomes:

$$\rho' = \frac{\sum_{n=1}^{N'} (r'_n - \bar{r}')(p'_n - \bar{p}')}{\sqrt{\sum_{n=1}^{N'} (r'_n - \bar{r}')^2} \times \sqrt{\sum_{n=1}^{N'} (p'_n - \bar{p}')^2}} \tag{37}$$

With N' the number of items rated by either one or the other actor, r'_n the score of one actor and p'_n the score of the other. $\bar{r}'$ and $\bar{p}'$ are the scores mean of respectively each actor.

The scores are computed using a score propagation up to depth $k = 1$, *i. e.* the direct friends only.

Using the extended similarity in Epinions dataset, an actor has then 4 similarity computable out of 5 friends, *i. e.* about 4 times more similarity computed.

6.2.2 *Evaluation*

In order to evaluate the extended similarity heuristic, we have run it with a score propagation depth $k \in \{1, 2, 3\}$ with the alchemy dataset, without correlation (*aka.* "NoCorrelation"), with the classical Pearson correlation coefficient (*aka.* "Correlation") and with the extended Pearson correlation coefficient (*aka.* "ExtendedCorrelation"). Since we use default similarity when the correlation is not computable, the coverage is the same for all.

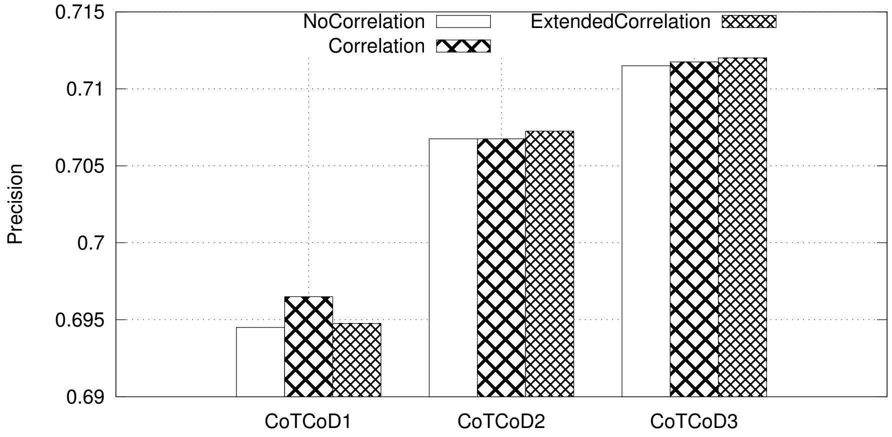


Figure 25: Influence of correlation on precision using Alchemy

Figure 25 shows that extended correlation does not really influence the precision on the prediction. However, actors manage to compute similarity for most of their friends, reducing the use of a constant as default similarity. Moreover this weight is used in other computations, such as confidence, *c.f.* section 4.2 page 47, or heuristics based on weight, *c.f.* section 6.5.3 page 96. Section 6.5.3 shows that extended similarity enhances weight influence on predictions precision.

6.3 RELATIVE SCORING

The k–depth social scorer computes absolute scores, *i.e.* it returns scores computed using actor or friends ratings. It does not take actors rating behaviour into account.

The following heuristic computes relative scores *i.e.* the difference between ratings and friends' ratings mean. Friends relative scores are aggregated and added to the actor's ratings mean in order to compute the final absolute score.

This approach has two main advantages. Firstly, it does not transmit absolute scores, but only relative scores: this means less information and therefore more privacy. Secondly, it takes into account actors rating behaviour as explained in section 2.2 eq.5, *i.e.* does an actor rate an average item with a rather low or high rating in general.

But this approach has a main drawback: if an actor has rated no item, no mean is available. In this case, 0.5 is used as a default mean. More generally, we consider that approach ineffective for cold start users, since they do not have enough ratings to provide a significant mean thus a significant rating behaviour.

6.3.1 *Relative score propagation*

Equation 38 represents the score propagation using relative scores instead of absolute scores. It only describes intermediate scores computations, the final computation made by the original requester is shown in equation 39.

No default scoring is considered here, but both approaches are compatible.

$$\delta s_k(a,i) = \begin{cases} r_{a,i} - \overline{r_a} & \text{if } \exists r_{a,i} \\ \dfrac{\sum_{f \in \mathcal{F}^{k-1}_{a,i,\omega}} \omega_{a,f} \times \delta s_{k-1}(f,i)}{\sum_{f \in \mathcal{F}^{k-1}_{a,i,\omega}} \omega_{a,f}} & \text{if } \nexists r_{a,i} \wedge \mathcal{F}^{k-1}_{a,i,\omega} \neq \emptyset \\ \perp & \text{otherwise} \end{cases} \tag{38}$$

The original requester adds his/her ratings mean to the intermediate relative score in order to compute the absolute score:

$$\Delta s_k(a,i) = \delta s_k(a,i) + \overline{r_a} \tag{39}$$

This propagation shares less information on friends ratings than the one defined in section 4.1.1.

Friends only share the difference between their ratings and their mean. They indicate that they relatively like an item because it has a better rating than their mean.

6.3.2 *Relative scoring evaluation*

In order to evaluate the relative scoring heuristic, we have run our scorer with $k \in \{1,2,3\}$, with or without relative scores.

We denote Absolute_k scorers returning absolute scores, as defined in section 4.1.1, equation 13, page 44. Relative_k are scorers returning relative scores, as defined above in equation 39.

Figure 26 shows side by side scorers with absolute or relative scores, for cold stat users, medium raters and heavy raters.

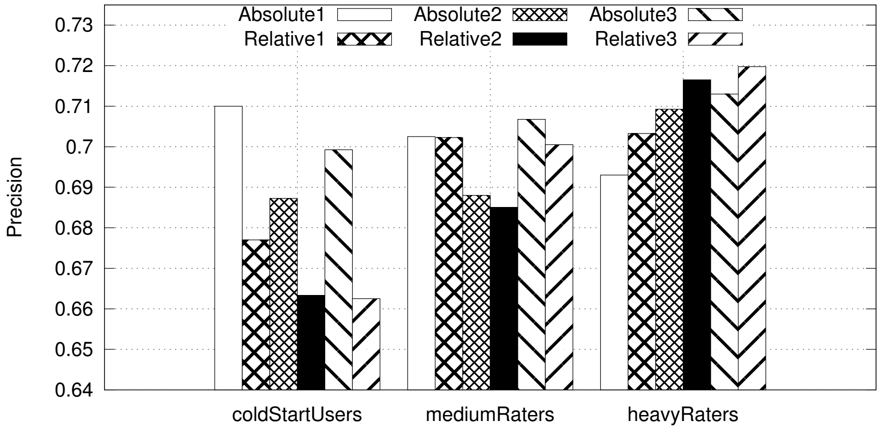


Figure 26: Absolute vs. Relative scores on Alchemy

It appears that relative scores are much less accurate than absolute ones for cold start users. This was expected since the rating behaviour of cold start users, represented by their mean, is not significant.

Medium raters see their predictions almost as accurate with or without relative scores. Relative scores has not impact on accuracy for those actors.

However, relative scores increase accuracy for heavy raters. For actors having more than 10 ratings, using relative scores is effective.

	$k = 1$	$k = 2$	$k = 3$
Absolute	1.222	1.173	1.154
Relative	1.205	1.167	1.165

Table 4: RMSE with relative or absolute score depending on k

Table 4 summarizes the RMSE for all actors. For $k \in \{1, 2\}$, the RMSE is lower with relative scores, therefore the precision is higher. However, with $k = 3$, the RMSE is higher with relative scores.

Globally, relative score heuristics does not improve that much the prediction accuracy. But it reduces the disclosure of information shared between friends, therefore enhancing privacy.

6.4 A NEW HOP

The score propagation described in chapter 4, section 4.1.1, page 43, stops when an actor has a rating to return or when the maximum depth propagation is reached.

This limits network flooding but also reduces the number of recommendations returned to the original requester. Moreover it explicitly returns an actor rating, thus revealing it. We introduce in this section an heuristic where actors propagate the score request even when they have a rating for that item. We call such heuristic *hopping* or score propagation with *hops*. Doing so, when an actor returns a rating, it is mixed with other ratings, therefore harder to recover.

6.4.1 *Score propagation with hops*

When an actor has a rating for an item, instead of ending the score propagation and returning his/her rating, the actor forwards the request. When his/her friends return scores, then the actor aggregates those scores with his/her own rating and returns the result.

The aggregation between the actor's rating and his/her friends' is made using a coefficient α, with $\alpha \in [0, 1]$.

Definition 32: Hop aggregation. *Let* $s_k(a, i)$ *be the score predicted using the scores from the friends of* a *on the item* i*. The hop aggregation is made with* $r_{a,i}$ *and* $s^F_k(a, i)$ *as follow:*

$$s_k(a, i) = \alpha \times r_{a,i} + (1 - \alpha) \times s^F_k(a, i)$$

Figure 27 shows the k–Depth Social Scoring example, as in section 4.1.1 page 43, but with hops. Scores between actors a_0, a_1 and a_6 differ in this figure compared to figure 8.

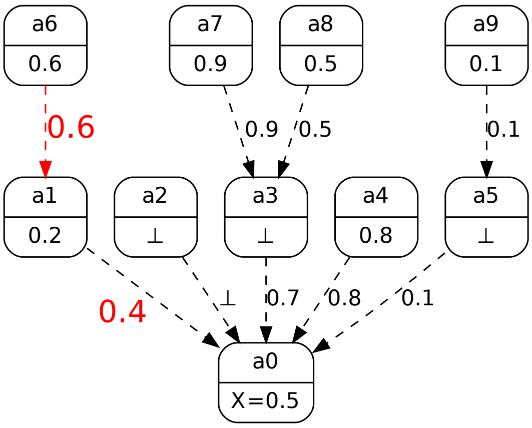
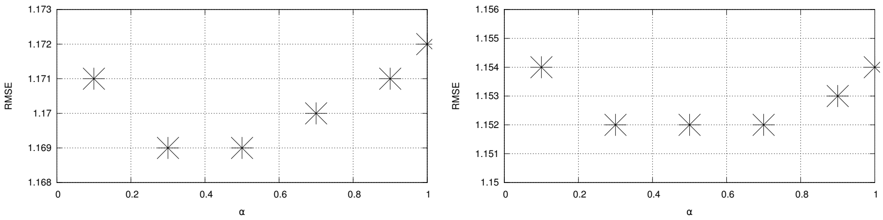


Figure 27: k–Depth Social Scoring with Hops; Example with $\alpha = 0.5$

Without hop, a_2 does not ask his/her friends since he/she has a rating for the item. With hops however, a_2 still asks for his/her friends scores, in this case a_6's score. Then, a_2 aggregates both scores: his/her own score 0.2 and a_6's score 0.6. Since $\alpha = 0.5$, the aggregate score is $0.5 \times 0.2 + (1 - 0.5) \times 0.6 = 0.4$. Therefore a_2 returns 0.4 to a_0 and the final predicted score is 0.5 instead of 0.45.

6.4.2 *Hops evaluation*

In order to evaluate the hopping heuristic, we have run our scorer with $k \in \{2, 3\}$. $k = 1$ produces the same results with or without hops, since the score propagation stops at the immediate friends. For each k, we have run the evaluation with $\alpha \in [0, 1]$ in order to see if it is better to strengthen the actor's rating or his/her friends' scores. The greater α, the more important the actor's rating in the aggregation. $\alpha = 1$ is equivalent to not using hop at all, since when an actor has a rating, the friends' scores are not taken into account at all ($1 - \alpha = 0$).



(a) $k = 2$

(b) $k = 3$

Figure 28: Influence of α on the RMSE, using hops

Figure 28 shows the RMSE of CoTCoD$_2$ and CoTCoD$_3$ with hops, depending on α. Since $\alpha = 1$ represents CoTCoDepth without hop, we can see that the hopping heuristic improves the precision in both cases[1].

In addition, both graphs point out that α optimized the precision around 0.5. Therefore doing a simple mean between an actor's rating and friends' scores is a good combination. When recommending to a friend, taking into account what we have been recommended by our friends seems to improve the recommendation.

6.5 FRIENDS SELECTION

In order to reduce the network overload caused by the multiplication of requests, this section introduces some heuristics that may be used in situation where the system need a low resources allocation on recommendation.

Since asking all friends is costly for an actor, in terms of time and resources, the following heuristics propose to select a subset of the actor's friends and ask only that group of actors in order to predict a score.

Indeed in our approach, actors ask all their friends in order to predict a rating for an item. In real life users do not wait for a recommendation from all their friends to take a decision and select an item. Usually they ask to a subset of their friends, based on acquaintance, trust, similarity and/or expertise.

Section 6.5.1 describes an heuristic where friends are randomly chosen before propagating the request to them, and only to them. Section 6.5.2 is an evolution of the previous selection, where friends are still randomly chosen, but only friends with a score on the item count. Section 6.5.3 shows an heuristic where actors ask the friends with the highest weights in priority.

---

[1] The RMSE difference between 1.150 in section 5.5.2 and 1.154 here comes from the evaluation. In this section, we use a 90% training set evaluation whereas in section 5.5.2 we use a leave-one-out evaluation.

6.5.1 *Random friends selection*

This heuristic randomly selects a subset of size p of the friends F_a for a given actor a and asks their scores. p, a positive integer, is a parameter of the heuristic.

6.5.1.1 *Selection*

During the score propagation, an actor a randomly selects a set of p actors among his/her friends. If he/she has less than p friends, all friends are selected. Then the actor propagates the score request only to them.

That is, the actor randomly selects a set S such than:

$$S = \begin{cases} S^* \mid S^* \subseteq F_a \ \wedge \ |S^*| = p & \text{if } |F_a| \geqslant p \\ F_a & \text{otherwise} \end{cases}$$

Once the selection is made, the actor propagates the request to all friends in S. If no one returns a score, then the actor cannot predict a score and returns $\perp$, no new selection is made.

6.5.1.2 *Evaluation*

In order to evaluate this selection, we have run the heuristics with $p \in [\![1, 30]\!]$ and with a score propagation depth k:

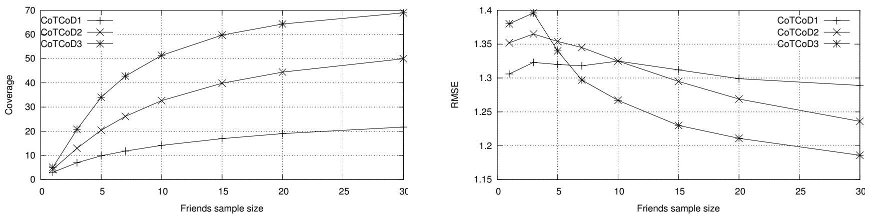
– $k \in \{1, 2, 3\}$ using Alchemy in figure 29 and

– $k \in \{1, 2\}$ using Flixster in figure 30.

Figures 29a and 30a show the coverage of the scores predicted depending on the size of the friends sample selection. Figures 29b and 30b show the RMSE of the predicted scores depending on the size of the friends sample selection. Figures 29c and 30c show the average duration needed to predict a score with a given k depending on the size of the friends sample selection.

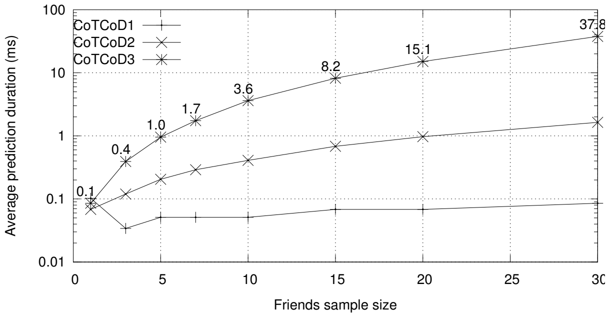
As we can see in figures 29 and 30, the more friends an actor asks, the higher coverage, precision and execution time.

Figures 29a and 30a outline that asking not enough friends will result in an low coverage. Asking less than 10 friends produces a low coverage preventing an adequate recommendation. Since friends are randomly selected, small samples causes few chances to get a score. From 20 friends, the coverage starts to increase less sharply, therefore it seems to be an interesting trade-off for a good coverage.



(a) Coverage

(b) RMSE



(c) Execution time

Figure 29: Random friends selection evaluation using Alchemy

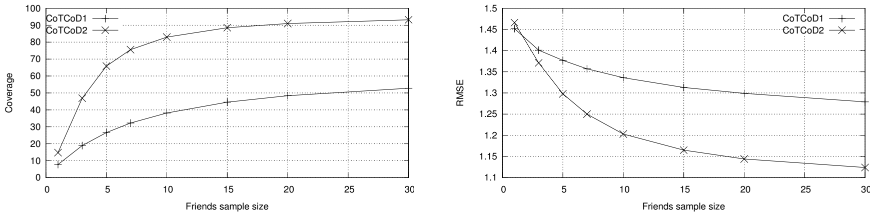
Figures 29b and 30b also illustrate a significant correlation between the size of the friends sample selection and the precision of the recommendation. With less than 10 friends, the coverage as shown in figures 29a and 30a is not high enough to provide good precision statistics. However the more friends selected, the lower the RMSE. Here again, 20 friends seems to be a good trade-off for a low RMSE, therefore a good precision.

Lastly, figures 29c and 30c illustrate the execution time of the evaluation depending on the friends sample size. The time scale is logarithmic which shows an execution time almost exponentially proportional to the size of the sample. Using Alchemy, the execution time of CoTCoD3 evaluation per score prediction is 15 milliseconds with 20 friends and it is 38 milliseconds with 30 friends, 2.5 times more.

This heuristics provides a good trade-off balance by selecting 20 friends for each actors. It provides a good coverage and a good precision, and it radically decreases the execution time and therefore the resources consumption.

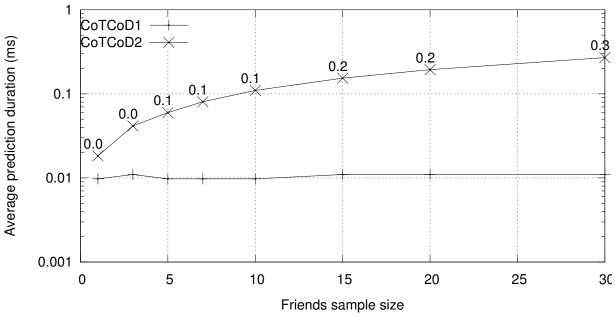
6.5.2 *Random raters selection*

In the previous section, friends are randomly selected and therefore the coverage suffers a lack of ratings since selected friends may or may not have a score to return. This section provides an heuristic verifying that the number of selected friends in the sample have all rated or predicted a score for the recommended item.



(a) Coverage

(b) RMSE



(c) Execution time

Figure 30: Random friends selection evaluation using Flixster

6.5.2.1 *Selection*

During the score propagation, an actor a selects randomly a first friend f with a strictly positive weight $\omega_{a,f}$ and asks his/her score. If the returned score is defined then the actor a selects another friend and starts this again. Until the number of defined scores is equal to p or there is no friends any more. Then the sample selection is over and the actor a may compute his/her own score using the returned friends scores.

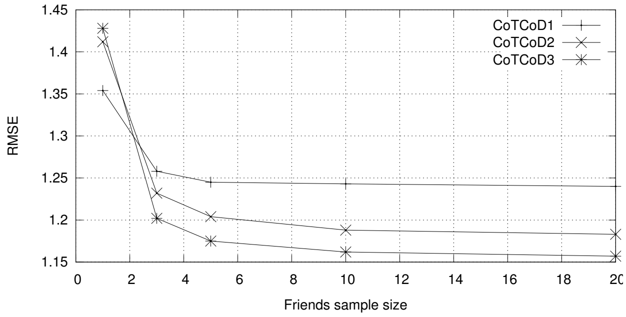
Since the selection is made by sending requests and waiting for results, no more step is needed in this heuristic besides computing the score. If no one returns a score, then the actor cannot predict a score and returns $\perp$, in the same way as without any heuristics. However, one can see that since only friends with scores are selected, the coverage is strictly the same with this heuristic than without any heuristic, for any $p > 0$.

6.5.2.2 *Evaluation*

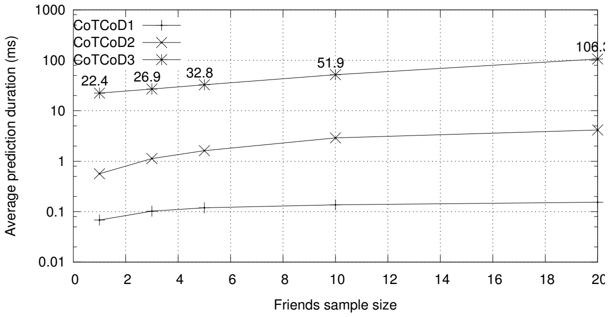
In order to evaluate the friends raters sample heuristic, we have run it with $p \in [\![1, 20]\!]$ and with a score propagation depth k:

- $k \in \{1, 2, 3\}$ using Alchemy dataset in figure 31 and
- $k \in \{1, 2\}$ using Flixster dataset in figure 32.

Since the coverage is not impacted by this heuristic, as explained above, no coverage figure is proposed. Figures 31a and 32a show the RMSE of the predicted scores depending on the size of the friends



(a) RMSE



(b) Execution time

Figure 31: Random raters selection evaluation using Alchemy

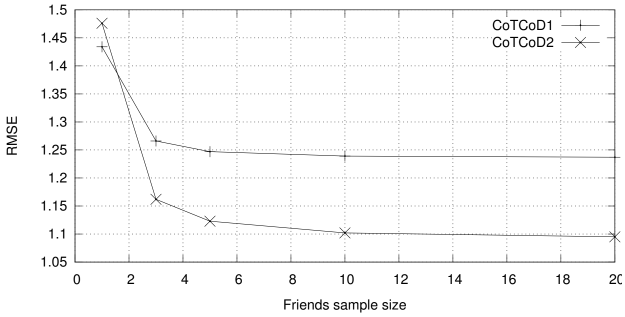
raters sample selection. Figures 31b and 32b show the average duration needed to predict a score with a given k depending on the size of the friends raters sample selection.

As we can see in figures 31 and 32, the more friends an actor asks, the better the precision and the lower the prediction time.

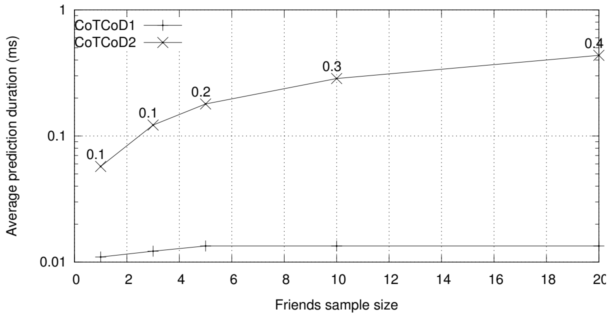
Figures 31a and 32a illustrate a significant RMSE decreasing between 1 and 5 friends raters. From 5 friends the error decreases slowly. Since the coverage is always the same, for any p, this shows that at least 5 scores are needed to provide an adequate recommendation. Anyhow the more selected friends, the lower the RMSE. Between 5 and 10 friends seems to be a good trade-off for a low RMSE, therefore a good precision.

Lastly, figures 31b and 32b depict the average duration needed to predict a score depending on the friends raters sample size. The time scale is logarithmic which shows a prediction time exponentially proportional to the size of the sample. Using Alchemy, the execution time of CoTCoD3 evaluation per score prediction is 33 milliseconds with 5 friends and 106 milliseconds with 20 friends, almost 3 times more.

This heuristics provides a good trade-off balance by selecting between 5 and 10 friends raters for each actors. It provides the best coverage and a good precision, and it radically decreases the execution time and therefore the resources consumption.



(a) RMSE



(b) Execution time

Figure 32: Random raters selection evaluation using Flixster

6.5.3 *Weight influence*

In the previous heuristics, the selection is made randomly, without taking into account the weights of the friends. Here, this weight is used to order friends and to select firstly the friends with the higher weights. This heuristic does not change the average prediction time compared to the previous one but try to improve accuracy for small samples, when weight matters.

6.5.3.1 *Selection*

During the score propagation, an actor a first gets the weights between him/her and all his/her friends. Then the actor selects the friend with the highest weight and asks his/her score. Then the actor a selects the second friend with the highest weight and starts this again. That is, until the number of defined scores is equal to p or there is no friend any more. At this point the sample selection is over and the actor a may compute his/her own score using the returned friends scores.

As in section 6.5.2.1, since the selection is made by sending requests and waiting results, no more step is needed in this heuristic besides computing the score. If no one returns a score, the actor cannot predict a score and returns $\perp$, in the same way as without any heuristics. However, one can see that since only friends with scores are counted, the coverage is strictly the same with this heuristic than without any heuristic, for any $p > 0$.

6.5.3.2 *Evaluation*

In order to evaluate the friends raters sample heuristic, we have run it with $p \in [\![1, 20]\!]$ and with a score propagation depth $k \in \{1, 2\}$.

We first used this heuristic with local similarity, but it did not change anything compared to the friends raters sample, defined in section 6.5.2. This is mainly due to the fact that the trust is always 1 and the local similarity is barely computable in this dataset, since an actor has an average of 8 friends and the intersection between common raters of an item and friends is almost always empty.

Therefore, this evaluation uses extended similarity in order to define a different weight for each friends. This also means that results may differ from previous evaluations, where no similarity has been used.

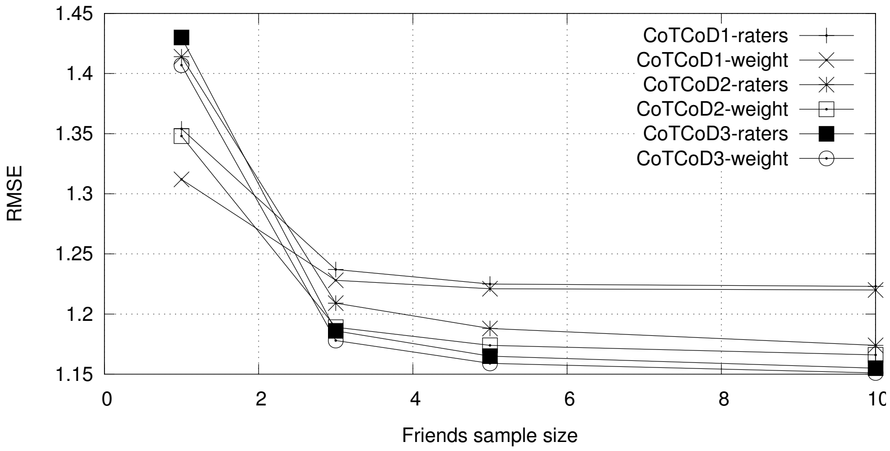


Figure 33: Weight influence compared to random raters selection

Figure 33 shows both random raters selection (CoTCoDk-raters) and weight influence (CoTCoDk-weight) heuristics with $k \in \{1, 2\}$. For low values of p, weight influences significantly the accuracy of the approach. Selecting similar friends is efficient for small selections. However, the greater p, the less weight influences the prediction. Randomly selecting friends without considering weight is effective if there are enough friends in the selection. This erases the disparity of the prediction and improve globally the accuracy.

This heuristic should be more efficient with disparate weights between friends, which will be more likely the case with wide social network, such as the Facebook one, where users have an average of 300+ friends. Unfortunately we do not have such a dataset to confirm that hypothesis.

6.5.4 *Conclusion*

In this section, we have seen three heuristics aiming at accelerating the prediction while maintaining an acceptable coverage and a good precision.

The first one, random friends selection, selects randomly friends for a given actor. It shows fast prediction but suffer a low coverage.

The second one, random raters selection, selects randomly friends who return a score for the given item. It is slower than the previous one but produces the highest coverage possible. Moreover, the precision is almost as good as without any heuristic, with a faster prediction.

The last one, weight influence, selects friends taking into account their weights. It is effective with small selections, *i.e.* less than five friends. For bigger selections, the gain is minimized by the amount of recommendation coming from selected friends.

In a peer-to-peer architecture, waiting the response from one peer can be costly, therefore the second and third approaches, random raters selection and weight influence, are not well suited. Indeed those heuristics select the sample by asking friends one at a time, waiting to see if the returned score is defined or not in order to take into account that score in the prediction.

In order to counter this drawback, one can adapt these heuristics with batches. An actor selects several friends instead of selecting only one friend, he/she then asks those friends for their scores and waits for an answer. If the number of computed scores is lower than p, then the actor selects other friends, otherwise the selection is over. If the number of scores is greater than p, the actor can use all of them, it will enhanced the precision.

6.6 EXPERTISE

In this section, we propose two heuristics that take into account the expertise of the actors during the score propagation.

In order to evaluate those heuristics, we need a dataset providing expertise from actors to item categories. Unfortunately, Flixster and Alchemy do not provide those data. But our own extraction of the Epinions website provides those pieces of information. Therefore we have used RED, described in appendix section B.1 page 120.

6.6.1 *Friends expertise*

The friends expertise heuristic takes into account the expertise of an actor's friends during the friends selection.

The friends selection is processed as follow: an actor asks only friends who are experts in the item category. If there is no expert, the actor asks the other friends as usual.

Selecting only experts when available has two main advantages. The first one is that an actor will not ask all his/her friends when he/she knows an expert in the category, therefore limiting the network resources consumption. The second one is that an expert who happens to be trusted by the actor should return a relevant score, therefore enhancing the accuracy.

Finally, this heuristic is compatible with the previous ones. If no expert is selected, the actor may use the *friends raters sample* heuristic to predict a score, for example.

6.6.2 *Global expertise*

The global expertise heuristic takes into account recommendation from all experts, even the ones that are not friends. This heuristic makes a strong assumption which is that experts let their profiles public and accept to share their recommendations to anyone.

The "friends" selection is processed as follow: for a given item category, an actor asks to all experts from this category and all his/her friends.

This heuristic virtually create links between all actors and experts. Therefore experts recommendations are taken into account by all actors.

Since there are more links in the network, this makes the whole algorithm use more resources. But it should also improve the coverage.

6.6.3 *Expertise evaluation*

In order to evaluate the expertise heuristics, we have run them with a score propagation depth $k \in \{1, 2, 3\}$.

Table 5 shows the RMSE, the coverage and the average duration in milliseconds for a rating prediction with each heuristic (*friends* and *global* expertise) and without any (*none*).

As expected, the coverage is a little bit lower with the friends expertise, since an actor selects only experts if any among his/her friends, and a little bit higher using global expertise, since an actor selects experts in addition to friends. But the impact on the coverage is not really significant.

However there is a slightly reducing on the average rating prediction time with the friends expertise: the prediction is about 15 % faster with that heuristic that without. All actors are not selected, which explains that improvement. Experts are few but they made a lot of ratings and have many relations. It is enough to limit the score prop-

Depth	Expertise	RMSE	Cov.	Time (ms)
$k = 1$	None	1.119	40.92	0.44
	Friends	1.122	36.52	0.90
	Global	1.133	43.81	0.90
$k = 2$	None	1.148	59.69	3.9
	Friends	1.147	53.52	3.2
	Global	1.157	61.85	7.1
$k = 3$	None	1.151	65.03	503
	Friends	1.150	58.50	425
	Global	1.160	66.73	1025

Table 5: Results for expertise heuristics on Epinions

agation by asking them in priority, and not others friends, without degrading accuracy.

The third column shows that none of the heuristics really improves the precision. It is almost not impacted by the friends expertise heuristic and the precision is worse with the global expertise than without any. This confirms our main assumption that friends provide more personalized recommendations than anonymous experts.

6.7 CONCLUSION

This chapter introduces three main heuristics.

The first one uses *hops* in order to reach all actors in the social network, up to depth k, in order to aggregate more ratings and improve the final recommendation accuracy. It keeps the algorithm purely local, unlike MoleTrust [MA07a], since no new relations are added in the social network, and still manage to collect all ratings.

The second one selects only a subset of friends in order to speed up the score prediction, without penalizing the accuracy of the result. It is perfectly suited to peer-to-peer architectures and focuses on resources consumptions optimizations.

The last one takes into account actors expertise in the recommendation. It shows no much interest for the recommendation with our datasets. But it should be interesting to evaluate it with other datasets, containing more data on expertise than our Epinions dataset.

Part IV

CONCLUSION

7

CONCLUSION

To the person who does not know where he wants to go there is no favorable wind.
— *Seneca*

In this thesis, we propose a recommender system based on social relations between users. Our system aims at using social networks with trust-based techniques, but without propagating trust. We consider social relations as predefined and therefore not inferable by computers. This means that users should communicate and share their profiles only with direct friends regarding a social network. We define purely local versions of our approach (without default score or with actor mean rating as default score) and a global version (with item mean rating as default score) respecting our privacy concerns.

Based on those assumptions, we have presented a Correlative and Trust-based with Confidence k-Depth social recommender system, CoTCoDepth, using limited knowledge and based on explicitly user-defined social relations.

It can be deployed on the users' devices, does not need heavy off-line preprocessing. Its complexity depends on the number of friends of the user. Scores are propagated in a P2P manner. Each peer has knowledge of the local user's friends' scores, but cannot access friends-of-friends' data.

7.1 CONTRIBUTIONS

In order to predict ratings, users ask their friends their scores on items. Friends return their scores in the network step by step, until reaching the original requester. Scores are either a rating if a user has one, or an aggregation of friends' scores.

Since we do not want to flood the whole network, we define a **maximum depth propagation**, depending on the density of the network. With dense networks, such as Facebook, a maximum depth propagation of 2 offers good coverage and accuracy. With sparse networks, such as Epinions, a maximum depth propagation of 3 is necessary.

Relations are weighted by an **explicit local trust coefficient** and an **implicit local similarity coefficient**. Trust is explicitly defined by users and represents the user belief on the usefulness of his/her friends recommendations. Similarity is computed by the system in order to promote recommendation coming from friends with similar rating behaviours, therefore similar tastes. Unlike traditional approaches, **similarity is locally computed** between friends, and not between all users. Since similar users and friends are usually disjoint, we propose an **extended version of the similarity** computation that is able to compute similarity for 80 % of friends in average in our evaluation datasets. The 20 % remaining friends receive a default coefficient in order to participate in the recommendation even so.

Confidence coefficients are associated with scores during the propagation. Confidence represents the system belief on the accuracy of the score. Users can use confidence as an indicator of the probable usefulness of the recommendation.

Confidence integrates several aspects: links weights, number of aggregated scores, variance of the aggregated scores, users distance and score freshness. Confidence is aggregated at each step before being forwarded to the next peer.

Experimentations show that **our score propagation offers high coverage with dense datasets**, such as Flixster or Appolicious. However sparse datasets, such as the Epinions ones, suffer a lack of trust relations. They usually require deeper propagation, which is not optimal in P2P architectures. In order to provide recommendation without propagating scores too deeply in those networks, we introduce **default scoring**. Default scoring is commonly used to deal with sparse datasets. We have adapted it to our architecture.

When a rating is not predictable with our propagation, a peer can either return nothing or return a default score. The probability of returning a default score is low in order not to pollute genuine scores. Default scores are propagated in the network as any other score but with a lower confidence. This lower confidence allows the aggregation to consider genuine scores with higher weight than default scores.

A default score is either the user's ratings mean or the item's ratings mean. The first strategy is purely local since a user only needs his/her own ratings to compute the mean. The second strategy is not purely local since a user needs to access the item's ratings mean, which requires centralized architecture. Nevertheless, that information is anonymous and does not reveal information on users, therefore this strategy only requires anonymous knowledge. We also propose to combine the two strategies in order to return the item's ratings mean if possible, or the user's ratings mean otherwise. Default scoring enhances our score propagation with sparse datasets.

We have evaluated our approach with **three datasets**: Flixster, Appolicious and Epinions Alchemy datasets. A maximum depth propagation of 2 provides high coverage with the dense datasets Flixster and Appolicious (more than 95 %). With sparse datasets, such as Alchemy, we need to propagate up to depth 3 and use default scoring in order to predict more than 95 % ratings. Using the Alchemy dataset, we have compared our approach with classical collaborative filtering (UserBasedCF and ItemBasedCF) and trust-based approaches (MoleTrust, RandomWalk and TrustWalker) regarding three metrics: coverage, precision and f-measure.

The results show that **our approach has the highest f-measure for all users and cold start users** with this dataset. This is mainly due to a high coverage, thanks to our default scoring strategies.

Our CoTCoDepth scorers have the **best precision for cold start users** (*i. e.* users that have rated few items) and a **high coverage even for users that have few (less than five) connections** with others. The coverage results are 90 % ratings for all users and almost 100 % ratings for users with more than four connections, despite a sparse dataset. To sum up, **our approach shows good results even for users that are weakly connected and/or when they did not rate many items, whereas they are usually the hardest users to recommend**.

Furthermore, we have defined **several heuristics pluggable in our system**. Some heuristics aim at **improving accuracy**, to the cost of more requests on the network, others **reduce the network consumption** by limiting requests propagation.

The **hopping propagation** allows users to ask their friends even when they already have a defined rating. This heuristic propagates scores from all friends up to the maximum depth propagation, thereby increasing the number of requests on the network. However it also increases the number of scores aggregated during the propagation and improves predictions accuracy.

Relative scoring takes into account users' rating behaviour by integrating their ratings mean as a bias. It also limits ratings disclosure since only relative ratings are propagated. Our evaluation shows that this heuristic improves accuracy for heavy raters, *i. e.* users with 10 or more ratings, has almost not impact on accuracy for medium raters, *i. e.* users with more than 4 ratings and less than 10 ratings, and reduces accuracy for cold start users, *i. e.* users with 4 ratings or less.

In order to adapt our approach to P2P architecture, we need to limit the number of requests on the network. Therefore, we have also focussed on **friends pruning** during the propagation. Here, a user selects only a subset of all his/her friends in order to propagate a request. This greatly reduces network overload and accelerates prediction. We show that selecting enough friends still provides accurate predictions. We first present a naive approach that selects randomly friends. Then we propose to consider the n first friends that return a score. Finally we take into account friends' weights in order to select the most similar friends first.

Our last heuristic concerns **users' expertise**. It tries to improve accuracy and to limit propagation by selecting in priority friends that are experts in the item category. We have also proposed a global heuristic, where users ask scores to experts known in the category, even if they are not friends. This heuristics tries to improve coverage as well as accuracy. Unfortunately, our dataset does not contain enough experts in order to measure the impact of this heuristic.

7.2 DISCUSSION

We have shown that a local vision of the ratings remains efficient in ratings prediction, with sparse or dense datasets. It shows good results even for weakly connected or cold start users, whereas they are usually the hardest users to recommend.

Chapter 5 compares our approach with others using Alchemy, a trust dataset. We have not implemented and evaluated TidalTrust since it propagates trust using the same kind of knowledge as MoleTrust but shows usually higher RMSE than MoleTrust [VCC11].

Compared to MoleTrust, our purely trust-based version CoTCoD3 suffers a lower precision with the same coverage. By definition, CoTCoDepth, MoleTrust and TidalTrust provide the same coverage. But MoleTrust aggregates more ratings to compute a score, since it builds extended users' web-of-trust with an extended-local knowledge on the network. This explains the small gain (less than 5 % lower error). However, our approach is purely local, is deployable on P2P architectures and does not add new relation in the network.

Trust-based approaches usually propagate up to depth 6 [MA07a, JE09], improving coverage and accuracy. Regarding our decentralization and privacy assumptions, we do not want to propagate that far in the network. But our algorithms can handle such a propagation. In centralized architectures, they could propagate scores up to depth 6. However our approach is the only one fully compatible with P2P architectures.

In order to counterbalance lower coverage due to lower depth propagation, we have introduced default scoring, adapted from classical default rating. Our default scoring proposition is actually compatible with existing trust-based approaches like MoleTrust or TidalTrust.

TrustWalker proposes to mix item-based collaborative filtering in trust-based recommendation. We interpret this as an evolved default scoring strategy. Instead of returning an item's rating, it returns another item's rating. Their good results prove that this makes more sense than returning for example the actor ratings mean as default score. However in order to compute item similarity as they do, they need a global access on all users ratings. This counters our assumptions on local knowledge and on privacy.

Some collaborative filtering systems also use default rating in order to counter sparse datasets. However those approaches are not local, which is why we have not implemented and evaluated them.

Nonetheless, our item mean rating default scoring strategy accesses items' ratings. This requires global knowledge. The main difference between this approach and the collaborative filtering ones resides in that to compute an item mean, we only need to access anonymous ratings and to aggregate them, without knowing who made which rating. A centralized peer could offer such a service by returning the mean without revealing ratings, using homomorphic encryption [HBC10].

The following table extends table 1 from section 2.5 by adding our approaches:

Approach	Graph	Knowledge	Computation	Relations
TextBasedCB	items similarity	local	heavy	implicit
UserBasedCF	users similarity	global	heavy	implicit
ItemBasedCF	items similarity	global	heavy	implicit
TidalTrust	trust network	extended-local	light	propagated
MoleTrust	trust network	extended-local	light	propagated
RandomWalk	trust network	local	light	explicit
TrustWalker	trust & items similarity	global	heavy	explicit
SocialMF	social network	global	heavy	propagated
Hoens et al.	social network	local	heavy	explicit
CoTCoDepth	social network	local	light	explicit
CoTCoDepth$_a$	social/trust network	local	light	explicit
CoTCoDepth$_{ia}$	social/trust network	anonymous	light	explicit

Table 6: Summary of recommender systems characteristics

With:

- CoTCoDepth: our main approach, without default score,
- CoTCoDepth$_a$: our approach using actor mean rating as default score,
- CoTCoDepth$_{ia}$: our approach using item mean rating or actor mean rating as default score if the former is not computable.

7.3 PERSPECTIVES

The pure local versions of CoTCoDepth are deployable on peer-to-peer architectures. We have already implemented them in a P2P simulator: PeerSim [MJ09], *c.f.* appendix C.2 page 138.

Our future work will focus on the evaluation of the network usage implied by our approach and simulate some P2P and decentralized constraints existing in such architectures (disconnection, dynamism, timeout, etc.). We will also observe the influence of our friends pruning heuristics.

During this work, we have defined magic constants such as the maximum depth propagation, the default similarity or the default scoring probability $P_{default}$. Those constants have been validated empirically but are dataset specific. It could be interesting to transform those constants into dynamic values.

Section 5.4 highlights that both maximum depth propagation and default scoring probability should depend on the network density. The more connections, the lower $P_{default}$ in order not to flood the network with default scores. Alongside, weakly connected users should propagate deeper than heavy connected ones, since they are less likely to reach a direct friend having a score to return.

In parallel, each user could use a default similarity based on his/her other computed similarity values. Extended similarity explained in section 6.2 reaches a 80 % success rate, therefore it makes sense to compute default similarity as the mean or the median of existing similarity values.

We have proposed a confidence coefficient in order to inform users on the recommendation reliability. However we did not evaluate the impact of this coefficient on recommendation. To do so, we could compute the correlation between the confidence and the error of the prediction. A correlation close to -1 indicates that a high confidence implies a low error. Unfortunately this requires some major modifications in our prototype.

The following step will be try confidence on existing approaches, such as classical collaborative filtering algorithms. Instead of using friends to compute confidence, we could use similar users.

Since sparse datasets are a real problem in trust-based recommender systems, we have integrated default scoring strategies. However default scoring is artificial. TrustWalker also proposes a default scoring strategy, using similar items ratings when a user has no rating. Regarding our knowledge assumptions, we cannot compute item-based similarity using correlation coefficients on ratings.

But our approach could integrate content-based items similarity. Content-based similarity values are often locally computed, since they depend on items themselves.

Our approach is compatible with some content-based systems presented in the state of the art, section 2.1, page 12. We could use specific similarity measures like in [PFW05] or semantic descriptions like in [DMO+12].

When a user does not have any rating to return, instead of computing default scoring as defined in section 4.3 page 55, he/she could return a similar item rating or the mean of similar items ratings.

Users would compute item similarity using items descriptions, such as movie genre, producer, actors, synopsis... and apply content-based approaches based on those descriptions, such as the TF-IDF measure explained in section 2.1.1, page 13.

We have proposed a system where users share data with their friends, and no one else. However since users propagate their scores in the social network, malicious users can disclose information on users ratings behaviours.

Social and trust relations assume that two friends would not attack each others. This is a strong assumption and secure recommender systems that take privacy into account should not rely only on it.

Hoens et al. well studied security concerns about social-based recommendation in [HBC10]. Their system encrypts ratings before transmitting them. Their approach requires heavy cryptographic algorithms and our propagation slightly differs since we also propagate confidence coefficients, therefore we cannot directly map their approach to ours. However it would be an interesting research to find a light cryptographic way to transmit and aggregate scores in the social network without revealing them.

Finally, another way to deal with privacy is to make users not propagating their own ratings. During the initialisation of this privacy protocol, friends would share a part of their ratings. Then, users would shuffle their ratings with their friends' and select a subpart of the result. This subpart would become their new profiles.

Once the initialisation finished, and for any score propagation, instead of returning their ratings, users would return either one of their ratings or one of their friends'.

This method would make tastes and preferences inferring much more difficult since an attacker would not know if the score is either the rating of the user, the rating of one of his/her friends, a genuine score computing with his/her friends or a default score.

Part V

APPENDIX

A

MANAGING CYCLES

When people agree with me I always feel that I must be wrong.
— Oscar Wilde

Contents

A.1 Cycles in score propagation 114
A.1.1 Ping-pong cycle 114
A.1.2 Loop 114
A.1.3 Duplicate scores 115
A.2 Extended formula 116
A.3 Evaluation of the extended formula 116

Figures

Figure 34 Cycle with 2 actors 114
Figure 35 Cycle with 3 actors 114
Figure 36 Cycle with 4 actors 116

Tables

Table 7 Parenting strategies evaluation 117

A.1 CYCLES IN SCORE PROPAGATION

By definition, cycles are present in any social network with at least one relation: "I am the friend of my friend". Indeed friendship is not oriented in our definition, therefore any two friends are together a cycle. But more complicated cycles occur in social networks: "the friend of my friend could be my friend".

Scores propagation must take cycles into consideration, in order to avoid unnecessary computation and network overload. In figures 34, 35 and 36, three cycles examples are shown in order to illustrate our approach.

A.1.1 *Ping-pong cycle*

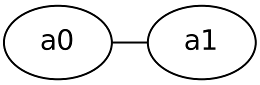


Figure 34: Cycle with 2 actors

Figure 34 shows a basic cycle in social networks, a simple friendship relation. If a_0 asks a_1's score, the latter may ask back a_0's score in order to compute his/her own score. This problem is simply avoided by not asking back the previous requester. This does not require any more information on the system nor on the request.

A.1.2 *Loop*

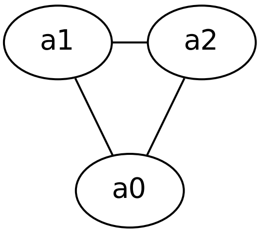


Figure 35: Cycle with 3 actors

Figure 35 shows a loop between three actors, each of them friends with the two others. For a score propagation of depth $k = 2$, no cycle will occur: a_0 asks a_1's score, who then may ask a_2's score, but since $k = 2$, a_2 will not ask anybody's score.

However, with $k \geqslant 3$, a_2 may ask a_0's score, despite a_0 being the original requester. We propose three solutions in order to counter this cycle.

A.1.2.1 *Unique id*

One way to solve this problem is to add an unique id to each request. When an actor receives a new request, he/she checks if it has not been processed before. If so, the request is ignored, otherwise it is processed and added to the request history.

There are two main drawbacks with this approach. The first one is the computational burden added by the request id history management. Actors must save request ids, but how long? When is a request considered as forgettable? The second one is the information leak risk. A malicious actor may broadcast requests in his/her network and analyze the results in order to scan the social network.

A.1.2.2 *Full requesters path*

In order to avoid that kind of cycles, the full requesters path can be added to the request. For example a_2 get a request from a_1 containing the previous requesters $\{a_0, a_1\}$. There is no additional computation with that approach since each peer does not have to memorize anything. However there are even more pieces of information revealed on the network, therefore it is not acceptable.

A.1.2.3 *Let it happen*

The last solution is to let cycles happen. Indeed the problem here is that a_0 will be asked for a score we know he/she does not have. In order to reduce that kind of situations, we decide that $k = 3$ is the maximum acceptable for a score propagation. With $k = 2$, no loop can occur; with $k = 3$ the original requester may be asked to answer his/her own request. But since he/she is the original requester, he/she does not have any score to return.

The additional cost is not excessive and the result will be the same anyway. Therefore we choose to let this scenario happen time to time in order to reduce information on the social network and to limit information leaks.

A.1.3 *Duplicate scores*

Figure 36 shows a situation where an actor answers several time for the same request. a_0 asks a score with $k = 3$. Let's assume that both a_1 and a_2 do not have a rating and have to ask their friends. Only a_3 provides a rating.

a_1 will ask a_2 and a_3 (not a_0 since he/she is the requester). Then a_3 will provide his/her own rating and a_2 will ask a_0 and a_3. So a_1 will get a_3's score twice, directly and indirectly, through a_2.

The same occurs when a_0 asks a_2's score.

We choose here again to let it happen. Since a_0 is friend with more actors, we choose to take his/her recommendation into account for

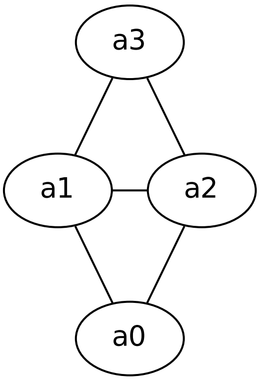


Figure 36: Cycle with 4 actors

each friend. Therefore this recommendation is more weighted than a usual one.

A.2 EXTENDED FORMULA

Our global strategy is therefore to limit cycles by not asking the requester and limit the depth propagation to 3. In other cases, we let duplicate and limited loops happen. The computational cost is not high (a few more requests), and actors with more connections are more influent, without knowing any global information on the social network. At no point an actor knows the original requester and/or previous requester besides the one asking him/her directly a score.

Definition 33:. *Let* $\mathcal{F}^k_{a,l,i,\omega}$ *be the set of* a*'s friends* f *without the previous requester* l *where* $s_k(f,i)$ *is defined and* $\omega_{a,f}$ *is not null:*

$$\mathcal{F}^k_{a,l,i,\omega} = \{f \in F_a \setminus \{l\} \mid s_k(f,i) \neq \perp \ \wedge \omega_{a,f} \neq 0\} \tag{40}$$

Therefore, the extended version of our approach is defined as:

$$s_k(a,l,i) = \begin{cases} r_{a,i} & \text{if } \exists r_{a,i} \\ \dfrac{\sum_{f \in \mathcal{F}^{k-1}_{a,l,i,\omega}} \omega_{a,f} \times s_{k-1}(f,a,i)}{\sum_{f \in \mathcal{F}^{k-1}_{a,l,i,\omega}} \omega_{a,f}} & \text{if } \nexists r_{a,i} \wedge \mathcal{F}^{k-1}_{a,l,i,\omega} \neq \emptyset \\ \perp & \text{otherwise} \end{cases} \tag{41}$$

By definition, $s_k(a,i) = s_k(a,a,i)$, for the original requester.

A.3 EVALUATION OF THE EXTENDED FORMULA

We have seen in section A.1 two ways to deal with cycles by knowing the direct previous requester or all previous requesters. We add another way which is not knowing anything and asking all our friends without knowing the previous requester.

There are therefore three strategies regarding cycles. The question is how much a requester knows about the previous requesters, *aka.* his/her parents. We call parent(s) the previous requester(s) in the score prediction.

ORPHAN the *orphan* strategy deals with cycles by not knowing any parent for a given actor

SINGLEPARENT the *single parent* strategy deals with cycles by knowing the direct parent of the request, *i. e.* the strategy choose in section A.1.2.3

ANCESTORS the *ancestors* strategy deals with cycles by knowing all previous parents of the request, from the original requester to the direct previous requester

We have implemented and evaluated the three strategies described above in order to deal with cycles. The coverage is not impacted by the strategies. Only the knowledge on the system and possibly the precision of the prediction. Finally, the number of involved actors is also affected since an actor included in a cycle will be asked several times for the same score.

Strategy	RMSE
Orphan	1.174
Single parent	1.175
Ancestors	1.175

Table 7: Parenting strategies evaluation

Surprisingly, table 7 shows that the precision is almost not impacted by the strategy. For a depth propagation of $k = 3$ with the 90 % training set, the RMSE equals 1.175 for the *single parent* and the *ancestors* strategies. However, the RMSE equals 1.174 for the *orphan* strategy. The difference is not significant therefore we cannot conclude anything on this.

However we can conclude that it is not necessary to know all ancestors to provide an accurate recommendation. Thus we can conceal the knowledge on the network to the previous requester at maximum, using the *single parent* strategy.

B

DATASETS

Some people go to priests others to poetry I to my friends.
— Oscar Wilde

Contents

B.1 Rich Epinions Dataset **120**
  B.1.1 Epinions dataset extraction 121
  B.1.2 Dataset structure 122
  B.1.3 Statistics 122
  B.1.4 Evaluation with this dataset 126
B.2 Appolicious dataset **126**
  B.2.1 Appolicious dataset extraction 127
  B.2.2 Dataset structure 127
  B.2.3 Statistics 128
B.3 Conclusion **129**

Figures

Figure 37 Database schema of the dataset 123
Figure 38 Trust distribution 124
  (a) Output trust count 124
  (b) Input trust count 124
Figure 39 Ratings count distribution 124
  (a) Users' 124
  (b) Items' 124
Figure 40 Database schema of the anonymised dataset 128
Figure 41 Following distribution 129
  (a) Output following count 129
  (b) Input following count 129
Figure 42 Ratings count distribution 129
  (a) Users' 129
  (b) Applications' 129

Tables

Table 8 Statistics depending on user characteristics 123
Table 9 Views distribution 125

Recommender systems evaluation is a difficult problem. One of the solution is given by offline evaluation, based on a dataset containing ratings that the system will try to predict. Therefore, datasets are really important in order to test and evaluate a system. However, some systems require information that is not always included in existing datasets.

Trust-based recommender systems need a trust network in order to provide recommendation. There is very few public datasets available. The main ones are Epinions and Flixster datasets, *c.f.* section 5.2 page 67. Those datasets contain ratings from users to items and a trust network. They are anonymised and no more information is available.

In this chapter, we describe two datasets that we have extracted during this thesis in order to provide datasets compatible with multiple approaches, content-based or collaborative filtering ones.

Section B.1 describes our Epinions datasets extracted in 2011. This dataset contains items description, categories and users description. It is referred as RED: a Rich Epinions Dataset [MGML11].

Section B.2 describes our Appolicious datasets extracted in 2012. Unlike the previous dataset, more textual information are provided in the Appolicious dataset, such as items description and reviews description. The social network included in Appolicious is based on followers, however users can automatically import Facebook friends into their network.

B.1 RICH EPINIONS DATASET

The Epinions[1] website contains reviews made by users on items. Items are any product or service. They have names and belong to one unique category. In a given category, items may show a common description structure. Categories are structured in a tree and may contain any number of items or subcategories.

Users build their web of trust within the community. A web of trust is a list of trusted or distrusted users. Anyone can trust or be trusted by anyone. Trusted users' reviews are promoted and distrusted users' reviews are less likely encountered. The web of trust may or may not be public, depending on the user settings.

A review contains a rating between 1 and 5 and a free text message. It may also contain some specific characteristics depending on the category (*e.g.* photo quality or shutter lag for cameras). Reviews can be commented and/or rated. A review rating is either "Not Helpful", "Somewhat Helpful", "Helpful", "Very Helpful", "Most Helpful" or "Off Topic". Express reviews are very short reviews that can only be tagged with "Show" or "Don't Show" whether they are valid or not.

---

[1] `http://www.epinions.com`

Epinions defines four kinds of users [2]:

- *Category leads* ensure high-quality review coverage of key items in their category and ensure that new reviews in their category are rated by a category lead or an advisor (see below).
- *Top reviewers* write high-quality reviews in their category of expertise. Their reviews have received the highest ratings from the Epinions community.
- *Advisors* rate reviews in their category.
- *Regular users* can review items, rate reviews and trust or block other users.

Orthogonally, any user can be a popular author: they are determined by the number of total visits to their reviews. Popular reviewers in specific categories are based on the users' total number of visits in that category. These users hold a top X rank (top 10, top 100...).

B.1.1 *Epinions dataset extraction*

Regarding the Epinions website structure, two strategies could have been used to do the extraction:

- extract all items, then for each item extract the relative reviews and the associated users
- extract all users, then for each user extract the relative reviews and the associated items

For the first strategy, Epinions proposes an easy way to browse items through items categories and subcategories. However there is no standardization between categories and parsing categories is not handy. Moreover, each category cannot show more than fifteen hundred items. And finally, many items do not have review, they are not useful regarding our purpose. We could also search all items through the search field with a dictionary approach. But the result list is also limited to fifteen hundred items.

We have then implemented the second strategy: search all users through the "members search" facility with a dictionary based approach. The fifteen hundred users limitation applies also here, but we have managed to extract a subsequent number of users with this approach: 240 000 users. Then, for each identified user, we have parsed his/her profile, reviews and web of trust, adding new users if any. This brought a total of about 307 000 users. For each users review, we have parsed the associated item if new and its category.

This approach ensures that items in the dataset have been reviewed at least once. However it does not ensure that each user has reviewed at least one item. We then cleaned the dataset by removing all unnecessary users, *i.e.* users with no trust relation nor review. Those users were found with the dictionary based approach and are certainly users who wanted to try Epinions or use a read only access.

---

[2] http://www99.epinions.com/help/faq/?show=faq_recognition

This extraction took two plain days of crawling in June 2011 on an Intel Core 2 Duo notebook with 3 Go of RAM.

We have encountered several problems during the dataset extraction. First of all, the Epinions website html structure is very particular, using a lot of table tags and very few CSS classes. This made the use of XPath very difficult. In addition, there are many exceptions in the pages structures, some pieces of information were missing sometimes whereas some others appeared not often. Moreover some special characters in users names were problematic. Categories breadcrumbs are not always consistent and made the category extraction pretty chaotic: we had to correct it manually.

B.1.2 *Dataset structure*

As shown in figure 37, the dataset is a relational database with the following tables:

- User: name (pseudo and profile url), location, top rank (may be null) and profile visits count
- Item: name, category and profile url
- Category: name, parent category, description url, lineage (path in the category tree) and depth (in the category tree)
- Review: a review associates a user with an item, it contains the rating, between 1 and 5, the review rating (mean of all review ratings associated with this review) and the review date
- Expertise: users who are experts in a category appear here with the expertise (category lead, top reviewer, advisor) associated with the considered category
- Trust: web of trust, *i.e.* a trust value (either -1 or 1) from one user to another, only positive trust values appear in the dataset
- Similarity: we have computed the similarity between all user pairs using the Pearson coefficient correlation [BHK98]. Since this operation may be long and is used in classical collaborative filtering, we provide it in order to ease recommendation; those values do not belong to the Epinions website

B.1.3 *Statistics*

The dataset contains 131 228 users, 317 755 items and 1 127 673 reviews, that is a 0.003 % density. 113 629 users have at least one rating. 47 522 users have at least one trust relation. 31 000 users have at

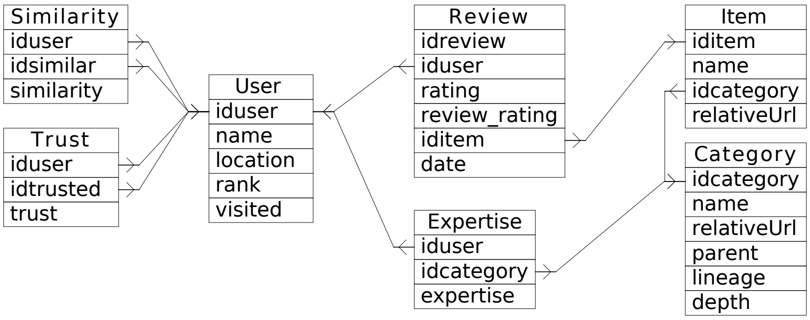

Similarity
iduser
idsimilar
similarity
Trust
iduser
idtrusted
trust
User
iduser
name
location
rank
visited
Review
idreview
iduser
rating
review_rating
iditem
date
Expertise
iduser
idcategory
expertise
Item
iditem
name
idcategory
relativeUrl
Category
idcategory
name
relativeUrl
parent
lineage
depth

Figure 37: Database schema of the dataset

least one similarity computed toward another user. 21 910 users have at least one review, one trust relation and one computed similarity. 4 287 users have neither reviews nor trust relation.

B.1.3.1 *Users and Trust*

Table 8 provides statistics on four users sets: all users, users with at least one review, users with at least one review and one trust relation and users with at least one review, one trust relation and one computed similarity. We provide for each set its cardinality and the average count of reviews, trust and similarity per user.

Users set	count	review	trust	similarity
all users	131 228	9	4	28
with review	113 629	10	4.5	32
with review and trust	34 410	25	15	95
with review, trust and similarity	21 910	38	20	149

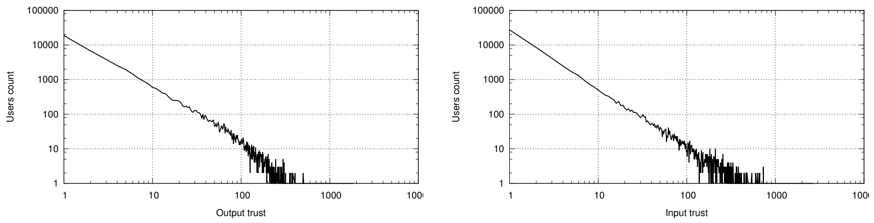
Table 8: Statistics depending on user characteristics

In average, a user has less than one trusted user with a computable similarity: intersection between trusted users and similar users is very small. However, experts have an average of 41 trusted users with a computable similarity.

The output and input trust are equally distributed and follow a power law (fig.38). This is common to main social network datasets. In average, users trust as many users as they are trusted.

B.1.3.2 *Categories and Expertise*

587 categories and sub-categories are provided. Among them, 21 root categories contain experts. 261 users are "experts", *i. e.* category leads, top reviewers or advisor in at least one category. Some of them have several expertises: the dataset contains 556 expertises. Only 261 experts in 131 228 users seem very low, but those experts made 488 217 reviews, *i. e.* almost half reviews. If we take experts with trust



(a) Output trust count

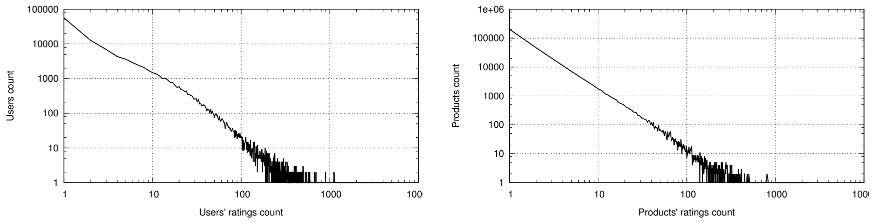
(b) Input trust count

Figure 38: Trust distribution

(respectively trust and similarity), they are 245 (resp. 241) and have made 463 991 (resp. 463 886) reviews.

B.1.3.3 *Ratings*

The ratings distribution is as follow: 7.2 % of 1, 7.4 % of 2, 12 % of 3, 30 % of 4 and 43.4 % of 5. We can see the particular distribution of the dataset. It is similar to the Trustlet [MA06] and Alchemy [RD02] datasets and seems to be the real distribution of the Epinions website.



(a) Users'

(b) Items'

Figure 39: Ratings count distribution

The ratings count distribution follows a power law (fig.39), a few users made a lot of ratings whereas most users made few ratings. Similarly, a few items has been reviewed many times whereas most items were reviewed a few times.

B.1.3.4 *Views*

In their evaluation, [MA07a] introduce views, *i. e.* parts of the dataset grouping particular users or items, that point out the advantages and drawbacks of the evaluated scorers regarding specific contexts. We have adapted them in table 9.

We define three categories of views for users and one category of view for items. Each category defines disjoint partitions of the users or items.

Users categories are not orthogonal: cold start users tend to be cold trusters and heavy raters tend to be heavy trusters.

Category		users/items %	ratings %
Rateness	No raters	13.4	0
	Cold start users	61.3	10.6
	Medium raters	15	15
	Heavy raters	10.3	74.4
Trustness	No trusters	62.6	17.8
	Cold start trusters	24.3	16.3
	Medium trusters	6.7	14
	Heavy trusters	6.4	51.9
Sheepness	Sheep	30.6	37.7
	Gray sheep	15.4	30
	Black sheep	54	32.3
Controversy	Unanimous	72.1	23.9
	Cold controversial	13.9	23.2
	Medium controversial	7.1	25.2
	Heavy controversial	6.9	27.7

Table 9: Views distribution

RATENESS The “rateness” category considers the number of ratings given by users.

NO RATERS provide no ratings.

COLD START USERS provide between 1 and 4 ratings.

MEDIUM RATERS provide between 5 and 15 ratings.

HEAVY RATERS provide more than 15 ratings.

TRUSTNESS The “trustness” category considers the number of trust values given by users.

NO TRUSTERS have no trust relations.

COLD TRUSTERS have between 1 and 4 trust relations.

MEDIUM TRUSTERS have between 5 and 15 trust relations.

HEAVY TRUSTERS have more than 15 trust relations.

SHEEPNESS The “sheepness” category considers users’ rating behaviours, it denotes the ability to rate more or less differently from the others: d is the average distance from users ratings to items mean (for each rated item), the bigger d, the more the actor gives ratings different from the majority.

SHEEP are users with $d \leqslant 0.5$.

GRAY SHEEP are users with $0.5 < d \leqslant 0.7$.

BLACK SHEEP are users with $d > 0.7$.

CONTROVERSIAL ITEMS The "controversy" category considers the standard deviation σ of items ratings.

UNANIMOUS ITEMS have $\sigma = 0$, all users rate them the same.

COLD CONTROVERSIAL are items with $0 < \sigma \leqslant 0.75$.

MEDIUM CONTROVERSIAL are items with $0.75 < \sigma \leqslant 1.1$. *Heavy controversial* are items with $1.1 < \sigma$.

We have balanced the two last categories regarding ratings ratio.

B.1.4 *Evaluation with this dataset*

As explained in section 5.1, the training set campaign splits the dataset into two disjoint sets: the training set and the evaluation set. Those sets need to be split randomly, which will influence the results. This evaluation must be made several times and its results aggregated, with the same training set size but with different shuffles.

In order to ease that evaluation, we have introduced the *ReviewEval* table. This table contains five random values for each review. Those five values *orderField1* to *orderField5* can be used to build five different evaluation shuffles. In order to run evaluation number one, one just needs to sort reviews with the first random value and to split the result set into the training and the evaluation dataset. Here is a sample SQL query in order to shuffle the dataset using the first random coefficient. One can build the 20 % training and 80 % evaluation dataset by appending "`LIMIT 225 534`" for the training dataset and "`LIMIT 225 534, 1 127 673`" for the evaluation dataset:

```
SELECT Review.* FROM ReviewEval
  INNER JOIN Review
  ON (ReviewEval.idreview = Review.idreview)
  ORDER BY orderField1;
```

B.2 APPOLICIOUS DATASET

The Appolicious[3] website contains reviews made by users on mobile applications. They have names, descriptions, prices, versions, sizes, publishers, compatible platforms and belong to one or more categories. Categories are not structured in subcategories, except for games categories which have a common parent category.

Users follow other users of the community. A user has follower and is followed by others. Anyone can follow or block another user.

A review contains a rating between 1 and 5, a free text message and a recommendation for some user categories. Reviews can be rated. A review rating is either "Not Helpful" or "Helpful".

---

[3] http://www.appolicious.com

Users have activity points, depending on their activity on the website. Writing reviews, building applications lists and so on provide activity points.

B.2.1 *Appolicious dataset extraction*

Regarding the Appolicious website structure, the same kind of strategies than for Epinions could have been used. However we did not want to build here an exhaustive dataset but we wanted to extracted a coherent subpart of the website, as connected as possible.

Therefore we have randomly chosen a user which was one of the top reviewers at the time. Then we have extracted and analyzed all her reviews, ratings and selected applications as well as her followers and following. Then we did that again for her social network.

This approach ensures that applications in the dataset have been reviewed at least once. However it does not ensure that each user has reviewed at least one application. We then cleaned the dataset by removing all unnecessary users, *i. e.* users with no following relation nor review. Those users are the leaf of the social network tree centered around the original user, without follower in our extraction and without review, therefore useless in recommendation.

This extraction took eight hours of crawling in February 2012 on an Intel Core i5 notebook with 4 Go of RAM allowed for the extraction.

This dataset extraction was easier than the Epinions' one. The html wax not XML compliant but we used TagSoup in order to use XPath. The structure of the source code then made the extraction quite straightforward.

B.2.2 *Dataset structure*

Figure 40 shows an anonymised version of the dataset. The complete dataset contains the following data:

- users: name, location, twitter account, personal website, date of registration, activity points, applifes, devices
- applifes: categories of users (books reader, gamer, etc.)
- devices: name of the device (model of mobile phone or tablet)
- followers: following link between users
- applications: name, description, price, date, version, size, publisher, categories, platforms
- categories: application categories
- platforms: OS compatible with the application
- ratings: rater (user), application rated, rating, date

  - reviews: reviewer (user), application reviewed, rating, date, review description, helpful, recommended applifes according to the reviewer

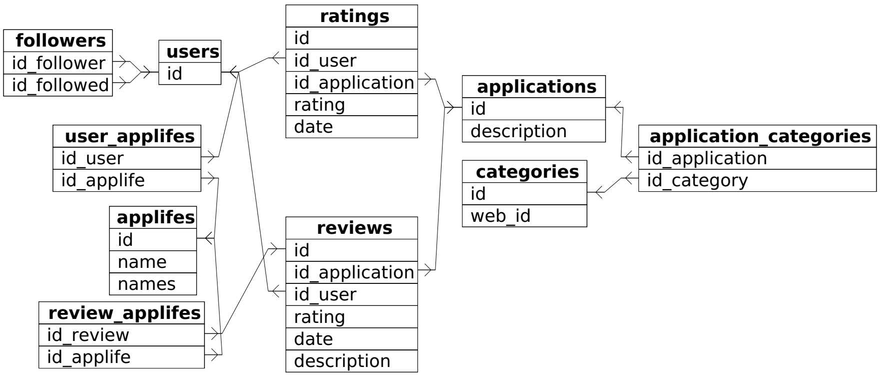

followers
id_follower
id_followed
users
id
ratings
id
id_user
id_application
rating
date
applications
id
description
application_categories
id_application
id_category
categories
id
web_id
user_applifes
id_user
id_applife
applifes
id
name
names
reviews
id
id_application
id_user
rating
date
description
review_applifes
id_review
id_applife

Figure 40: Database schema of the anonymised dataset

Applifes are some categories of users regarding their day to day behaviour and tastes. A user may define in his/her profile which applifes describe him/her the best. Then a reviewer may select which kind of applifes should like an application.

The applifes defined in the Appolicious website are enumerated in the following list:

- Animal Lover
- Book Reader
- Budget Hawk
- Career Person
- Film/TV Fan
- Foodie
- Gamer
- Music Fan
- News Junkie
- Outdoors Enthusiast
- Parent
- Road Warrior
- Shopaholic
- Social Butterfly
- Sports Fan
- Student
- Tech/Social Networking Junkie
- Workout Junkie

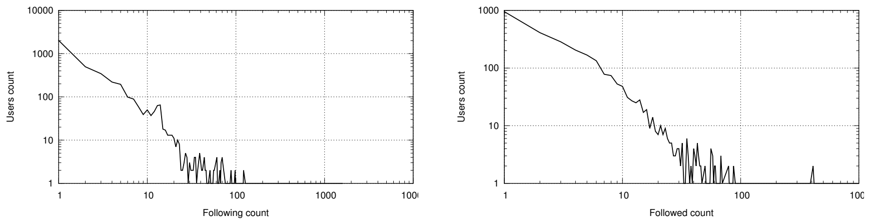
B.2.3 *Statistics*

The dataset contains 4 058 users, 8 935 applications, 28 963 ratings and 12 546 reviews, with 10 605 common ratings/reviews, that is a 0.08 % density. 1 007 users have at least one rating. All users follow at least one other user.

B.2.3.1 *Following*

There are 20 815 following links, that is 5 following/follower per user in average.

The output and input following links are equally distributed and follow a power law (fig.41). This is common to main social network datasets. In average, users follow as many users as they are followed.

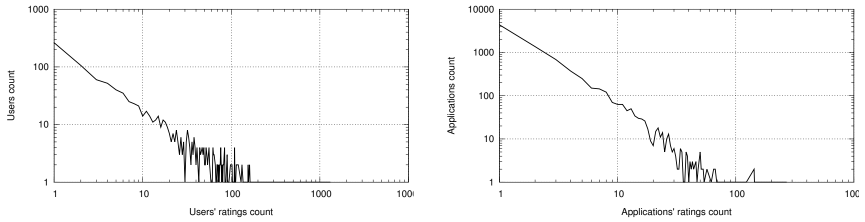


(a) Output following count (b) Input following count

Figure 41: Following distribution

B.2.3.2 *Ratings*

The ratings distribution is as follow: 2.5 % of 1, 5.1 % of 2, 20 % of 3, 37 % of 4 and 35.4 % of 5. The dataset is more equally distributed on the values 3, 4 and 5. However there are very few 1 and 2 ratings.



(a) Users' (b) Applications'

Figure 42: Ratings count distribution

The ratings count distribution follows a power law (fig.42), a few users made a lot of ratings whereas most users made few ratings. Similarly, a few items has been reviewed many times whereas most items were reviewed a few times.

B.3 CONCLUSION

This chapter presents two datasets extracted from the Epinions and the Appolicious website.

Existing datasets usually suffer a lack of cross-domain information. Ratings provide classical collaborative filtering the base to compute similarity coefficients. Trusts allow trust-based approaches to predict ratings. Items features, such as names, descriptions and categories, introduce content-based compliance. Moreover text reviews can be analyzed in order to improve the recommendation. Finally, information on users (applifes, location, twitter account) may be inferred to enhance the social network.

The Appolicious dataset contains much fewer users than the Rich Epinions Dataset, however it is also denser: 0.08% instead of 0.003% for RED.

Our datasets are therefore usable for the evaluation of many kinds of recommender systems. It aims at providing enough information for Content-Based, Collaborative Filtering and Trust-Based ones.

C

PROTOTYPES

The old believe everything, the middle-aged suspect everything, the young know everything.
— Oscar Wilde

Contents

c.1 Scars prototype 132
  c.1.1 Usage 132
  c.1.2 Modules 133
  c.1.3 Scorer 134
  c.1.4 Evaluation 136
  c.1.5 Conclusion 137
c.2 P2P prototype 138
  c.2.1 Implementation 138
  c.2.2 Results 138
  c.2.3 Conclusion 139

During this thesis, we have designed and implemented two different prototypes:

- "Scars" is used to evaluate our approach and compare it with algorithms of the literature (section C.1),
- the second prototype simulates our approach in a peer-to-peer architecture (section C.2).

C.1 SCARS PROTOTYPE

"Scars", for Scala recommender systems, is a framework we have implemented in order to evaluate recommender systems. There exist several recommender system frameworks released on the Internet:

- Apache Mahoot[1] is a machine learning library. It contains implementations of collaborative filtering, k–means clustering, singular value decomposition, etc.
- Twitter released recently its own framework Scalding[2]. It eases writing of MapReduce jobs in Hadoop[3] with natural code in Scala.
- LensKit[4] proposes a framework to develop or evaluate recommender systems.
- EasyRec[5] is an open-source recommendation engine to add recommendation in existing websites.

Some of them allow to build recommender systems whereas others are also evaluation frameworks.

However we have developed our own framework because we wanted a pure Scala framework:

- optimized to our multi core server,
- that can handle our specific datasets defined in appendix B,
- compatible with our approach and
- computing various statistics on results.

C.1.1 *Usage*

Our evaluation framework is command line only. We have written a script in shell in order to run the framework through maven.

Listing 1 shows our framework usage. The four first parameters are exclusive and select the evaluation dataset. `epinions`, `appolicious` and `flixster` load the appropriate dataset from text files. `database` is used to load RED, *c.f.* section B.1, from a database.

---

[1] `http://mahout.apache.org`
[2] `https://github.com/twitter/scalding`
[3] `http://hadoop.apache.org`
[4] `http://lenskit.grouplens.org`
[5] `http://www.easyrec.org`

Listing 1: Scars usage

```
USAGE:
  --epinions DATASET_PATH
  --appolicious DATASET_PATH
  --flixster DATASET_PATH
  --database DATASET_URL

OPTIONS:
  --save SAVE_DIR
  --load LOAD_DIR
  --reload
  --view
  --metrics LIST_OF_METRICS
```

The `save` option lets the framework save all predicted ratings in a text file in order to reuse the results. Since our approach implementation uses mixins traits in Scala, it is not yet possible to load the evaluated scorers list from a file. This list must be hard-coded. However, the `load` option loads predictions from previously saved text files instead of predicting ratings with scorers. The `reload` option continues previously stopped evaluation by reloading intermediate files in order not to compute again already predicted scores.

The `view` option computes and shows the metrics for each view implemented in the system. Views are defined in section B.1.3.4, page 124.

Finally, the `metrics` option selects which metrics we want to use to evaluate the scorers. `LIST_OF_METRICS` represents a coma separated list of metrics names, defined in section C.1.4.2.

C.1.2 *Modules*

Our framework contains the following modules:

1. Core: contains API and specific mathematics classes,
2. Scorer: contains recommender systems implementations, *aka.* scorers,
3. Evaluation: handles the evaluation part and is runnable,
4. FileDataset: loads file datasets,
5. DBDataset: loads database datasets,
6. Utile: contains generic classes used by other modules.

We detail the most important parts in the following. The Core module contains our framework API. The main interfaces (*i. e.* traits in Scala) are:

- `Actor`: An actor represents a user or a peer. It contains a set of reviews, a set of friends and a set of similar actors.
- `Item`: An item contains a set of reviews and a set of similar items.

  - `Review`: A review contains a rating from an actor to an item, with optionally a date.
  - `Scorer`: A scorer is a recommender algorithm that optionally return a score for a tuple (actor, item).
  - `Score`: A score contains a predicted rating and the confidence associated with the prediction.

The complete module contains more pieces of information, depending on the dataset used for the evaluation: item's description, item's category, actor's expertise, actor's rank and review's helpfulness. The Appolicious dataset contains even more data, that we have not implemented yet.

C.1.3 *Scorer*

C.1.3.1 *Interfaces*

A scorer must implements one of the `Scorer` (listing 2), `ActorScorer` (listing 3) or `ItemScorer` (listing 4) traits.

Listing 2: Scorer trait

```
trait Scorer {

  abstract def score(actor: Actor, item: Item, without: Set[Review]):
      Option[Score]

  def rating(actor: Actor, item: Item, without: Set[Review]): Option[
      Score] = {
    for {
      review <- actor.review(item) if !(without.contains(review))
    } yield Score(review.rating)
  }

}
```

`Scorer` is the main trait. Recommender system algorithms implement the `score` function. This function tries to predict a rating from the `actor` to the `item`.

By contract, it must not use reviews contained in the set `without`, those reviews are the one removed depending on the evaluation campaign, *c.f.* section 5.1, page 65. For example, the function `rating` returns the rating from the `actor` to the `item` if and only if this rating exists and is not contained in `without`.

Scorers that implement this trait predict ratings one by one, for each tuple (actor, item). They return `Option[Score]` which may be:

  - `None` if no score is predicted,
  - `Some(score)` if `score` has been predicted.

Listing 3: ActorScorer trait

```
trait ActorScorer extends Scorer {

  abstract def scores(actor: Actor, reviews: Set[Review], without: Set[
      Review]): Map[Review, Option[Score]]

}
```

`ActorScorer` optimizes algorithms that can factorise processes per actor, for example the ones that compute the extended web-of-trust for each actor.

The `scores` function takes one `actor` but a set of `reviews` (and their associated items). Algorithms must predict all ratings contained in the input `reviews`. For each prediction, the `without` reviews and the actual `review` must not be used.

Listing 4: ItemScorer trait

```
trait ItemScorer extends Scorer {

  abstract def scores(item: Item, reviews: Set[Review], without: Set[
      Review]): Map[Review, Option[Score]]

}
```

`ItemScorer` is the counterpart of `ActorScorer` for scorers that can optimize predictions for a given `item` with multiples `actors`.

C.1.3.2 *Implementations*

We have implemented several recommender systems during this thesis. Some are self dependant, some other can be mixed in with several traits[6].

Our main implementations are:

- `FixedScorer`: returns always the same rating, *e.g.* 4 is amazingly accurate on Epinions
- `UserBasedCF`: *c.f.* section 2.2.1
- `ItemBasedCF`: *c.f.* section 2.2.1
- `TidalTrust`: *c.f.* section 2.3.1
- `MoleTrust`: *c.f.* section 2.3.2
- `RandomWalk`: *c.f.* section 2.3.3
- `TrustWalker`: *c.f.* section 2.3.3
- `TrustAll`: *c.f.* section 5.4.1.1 page 73
- `CoTCoDepth`: our algorithm, *c.f.* chapter 4 page 41

---

[6] This feature is Scala specific, more details on `http://www.scala-lang.org/node/117`

We have also implemented some feature scorers, the followings take a list of scorers in argument:

- `CompositeScorer`: returns the first predicted score using all scorers one at a time
- `FilterScorer`: returns the last predicted score if all scorers have predicted something, otherwise return `None`

Finally, since ratings are integers most of the time, we have also implemented two classes that round the result of a given scorer:

- `LeveledScorer`: returns the nearest available rating, *e. g.* in Epinions, if the input scorer computes 2.3, this scorer returns 2.
- `SegmentedScorer`: returns an available rating with equiprobability for each rating. For example, in Epinions ratings are in $[\![1, 5]\!]$, the previous scorer returns 1 for each score in $[1, 1.5[$ and 2 for each score in $[1.5, 2.5[$: this is not equiprobable. The `SegmentedScorer` segments ratings in equiprobable classes. For Epinions, it returns 1 for each score in $[1, 1.8[$, 2 for each score in $[1.8, 2.6[$, . . . and 5 for each score in $[4.2, 5]$.

Those scorers artificially improve MAE (Mean Absolute Error), but degrade RMSE, since medium errors become no errors (2.4 becomes 2) or larger errors (2.6 becomes 3).

C.1.4 *Evaluation*

C.1.4.1 *Prediction*

Evaluation greatly takes advantage of parallel collections in Scala. Predictions are computed in parallel. To ease implementation, we ensure one scorer instance per prediction.

Depending on the `Scorer` trait implemented by the scorer (`Scorer`, `ActorScorer` or `ItemScorer`), the scorer tries to predict one score, one actor's scores or one item's scores. The prediction duration is automatically added to the score.

Once the scorers have predicted all scores, some statistics metrics are computed on the results.

C.1.4.2 *Metrics*

We have implemented the following metrics that can be used with the `--metrics` option (*c.f.* section 1):

- `total`: number of ratings to predict
- `cov`: coverage (in percent) of predicted ratings
- `mae`: Mean Absolute Error
- `rmse`: Root Mean Square Error
- `rae`: Relative Absolute Error

- `rrse`: Root Relative Square Error
- `wae`: Weighted Absolute Error
- `rwse`: Root Weighted Square Error
- `pcc`: Pearson Correlative Coefficient
- `srcc`: Spearman Rank Correlative Coefficient

Some additional metrics exist "per actor" that reduce heavy raters importance compared to cold start users:

- `atotal`: number of actors that have a rating to predict
- `acov`: average coverage (in percent) of predicted ratings per actor
- `amae`: average Mean Absolute Error per actor

Most of these metrics are classical but `wae` and `rwse`. `cov` is defined in eq. 33 and `rmse` is defined in eq.34, page 72.

We have defined `wae` and `rwse` in [MML11]. We use the same notation as in section 5.3.4, page 72:

$$WAE = \frac{\sum_{n=1}^{N} (|p_n - r_n| \times |r_n - \overline{r_i}|)}{N} \tag{42}$$

$$RWSE = \sqrt{\frac{\sum_{n=1}^{N} ((p_n - r_n) \times (r_n - \overline{r_i}))^2}{N}} \tag{43}$$

With $\overline{r_i}$ the mean of item i's ratings. Those metrics reduce error importance on ratings close to the items' ratings mean, for a given item. Those ratings are usually easy to predict, by computing a simple mean for example, thus less meaningful with respect to the evaluation.

C.1.5 *Conclusion*

The "Scars" prototype is a modular framework we have implemented in order to try our algorithms and to compare them with existing approaches. Scala, a high level language, allows to easily and quickly add new implementations. Its compatibility with Java is also advantage.

However this prototype does not take into account decentralization. It provides no representation of the architecture. That is why we have also implemented the next prototype in order to observe our approach in P2P architectures.

We have not released the source code yet, but you are welcome to send us an email for more information.

C.2 P2P PROTOTYPE

We have implemented our second prototype in a P2P simulator: PeerSim [MJ09], thanks to Emmanuel Guillot [GML11].

PeerSim takes care of the P2P architecture simulation, leaving algorithmic details to the programmer. We have used the cycle-based engine in order to simplify the development. But our approach is compatible with the event driven engine.

PeerSim takes a configuration file that contains network information: peers number, protocol, peer implementation class, neighborhood, etc.

C.2.1 *Implementation*

The class `Peer` represents a peer in the network, associated with one user. Each peer exchanges messages that can be a `Query` (for a score) or a `Reply` (a computed score).

For computational reasons, we do not exchange all queries at each cycle. We rather distribute all queries on several cycles in order to process some queries at a time.

The simulation initialisation first creates peers, assigns them their neighborhood and users' profiles (*i.e.* ratings). Then queries are instantiated and distributed on different cycles. There is one query per rating to predict. The peers list comes from the RED database, *c.f.* appendix B.1. The queries list comes from a text file.

Each cycles processes first the corresponding incoming queries. For each query, a peer returns its score if it has one, otherwise it generates new queries for its neighborhood. For each reply, if all neighborhood of a peer has replied, the peer computes the score and returns it. Otherwise it waits for the next replies. Then responses are returned.

The end occurs when all queries have been answered. At this point, all results are stored in a file, similarly to the `--save` option in "Scars".

C.2.2 *Results*

The dataset contains about 130 000 users for 1 100 100 ratings to predict. Therefore we had some scalability challenges. Simulations were run on Intel Core 2 Duo notebook with 1.5Go allocated memory.

The first evaluation took 9 minutes for a depth propagation of 1 with 500 cycles. But it did not handle a depth propagation of 2. Therefore we have optimized our prototype by merging queries between peers: if a peer has to ask ten scores to the same peer, it sends only one query with the ten items. The second evaluation took 28 minutes for a depth propagation of 2 with 2 000 cycles. We did not try a depth propagation of 3.

The score predictions were the same as with "Scars", validating our prototypes.

C.2.3 *Conclusion*

This prototype evaluates our approach in P2P architectures. Its implementation is not yet fully optimized but is functional.

We are planing to observe more criteria during the simulation, such as:

- the number of exchanged messages per peer per prediction,
- the average waiting time (or number of cycles) per peer per prediction,
- the network overload gain of our heuristics,
- the disconnections and reconnections impact on the prediction,
- etc.

Moreover we want to try and plug an existing phone as a peer in the prototype. PeerSim will simulate the other peers while the phone will ask scores to its friends. To do so, we have to implement a separate node acting as a proxy for the phone.

We have not released the source code yet, but you are welcome to send us an email for more information.

BIBLIOGRAPHY

[AT05] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749, 2005.

[BC92] N.J. Belkin and W.B. Croft. Information filtering and information retrieval: two sides of the same coin? *Communications of the ACM*, 35(12):29–38, December 1992.

[BHBL09] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[BHK98] J.S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th conference on Uncertainty in Artificial Intelligence*, pages 43–52. Madison: Morgan Kaufmann, 1998.

[BKG$^+$12] Y. Bachrach, M. Kosinski, T. Graepel, P. Kohli, and D. Stillwell. Personality and patterns of Facebook usage. *Proceedings of the 3rd Annual ACM Web Science Conference on - WebSci '12*, pages 24–32, 2012.

[BS97] M. Balabanović and Y. Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.

[BYRN99] R. Baeza-Yates and B. Ribeiro-Neto. *Modern information retrieval*. ACM press New York, 1999.

[CDHP11] M. Chevalier, A. Dattolo, G. Hubert, and E. Pitassi. Information Retrieval and Folksonomies together for Recommender Systems. In Christian Huemer and Thomas Setzer, editors, *ECommerce and Web Technologies*, volume 85 of *Lecture Notes in Business Information Processing*, pages 172–183. Springer Berlin Heidelberg, 2011.

[Deu62] M. Deutsch. Cooperation and trust: Some theoretical notes. In M. R. Jones, editor, *Nebraska Symposium on Motivation*, pages 275–319. Nebraska University Press, 1962.

[DMO$^+$12] T. Di Noia, R. Mirizzi, V.C. Ostuni, D. Romito, and M. Zanker. Linked open data to support content-based recommender systems. *Proceedings of the 8th International Conference on Semantic Systems*, pages 1–8, 2012.

[GH06] J. Golbeck and J. Hendler. FilmTrust: movie recommendations using trust in web-based social networks. In *CCNC 2006. 2006 3rd IEEE Consumer Communications and Networking Conference, 2006.*, volume 1, pages 282–286. IEEE, 2006.

[GML11] E. Guillot, S. Meyffret, and F. Laforest. Recommandation d'objets par relations de confiance dans les reseaux sociaux. Technical report, INSA de Lyon, Lyon, 2011.

[Gol90] L.R. Goldberg. An alternative "description of personality": The Big-Five factor structure. *Journal of Personality and Social Psychology*, 59(6):1216–1229, 1990.

[Gol05] J. Golbeck. *Computing and applying trust in web-based social networks*. PhD thesis, University of Maryland at College Park, 2005.

[Gol06] J. Golbeck. Combining provenance with trust in social networks for semantic web content filtering. *Provenance and Annotation of Data*, pages 101–108, 2006.

[Has10] O. Hasan. *Privacy Preserving Reputation Systems for Decentralized Environments*. PhD thesis, Institut National des Sciences Appliquées de Lyon, 2010.

[HBC10] T.R. Hoens, M. Blanton, and N. Chawla. A private and reliable recommendation system using a social network. In *Proceedings of the 2010 IEEE Second International Conference on Social Computing*, pages 816–825. IEEE Computer Society, 2010.

[HC10] J. He and W.W. Chu. A Social Network-Based Recommender System (SNRS). *Data Mining for Social Network Data*, 12:47–74, 2010.

[HKTR04] J.L. Herlocker, J.A. Konstan, L.G. Terveen, and J.T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[HWS09] C.W. Hang, Y. Wang, and M.P. Singh. Operators for propagating trust and their evaluation in social networks. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 1025–1032. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[JB06] M. Jelasity and O. Babaoglu. T-Man: Gossip-based overlay topology management. *Engineering SelfOrganising Systems*, 3910:1–15, 2006.

[JE09] M. Jamali and M. Ester. TrustWalker: a random walk model for combining trust-based and item-based recommendation. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 397–406. ACM, 2009.

[JE10] M. Jamali and M. Ester. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 135–142. ACM, 2010.

[JVG$^+$07] M. Jelasity, S. Voulgaris, R. Guerraoui, A.M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *ACM Transactions on Computer Systems*, 25(3):1–36, August 2007.

[KLMT10] A.M. Kermarrec, V. Leroy, A. Moin, and C. Thraves. Application of random walks to decentralized recommender systems. In Mohamed Lu, Chenyang and Masuzawa, Toshimitsu and Mosbah, editor, *Principles of Distributed Systems*, pages 48–63. Springer Berlin / Heidelberg, 2010.

[KT12] A.M. Kermarrec and F. Taïani. Diverging towards the common good. In *Proceedings of the Fifth Workshop on Social Network Systems*, pages 1–6, New York, New York, USA, 2012. ACM Press.

[LB09] D.H. Lee and P. Brusilovsky. Does Trust Influence Information Similarity? In *Proceedings of Workshop on Recommender Systems & the Social Web, the 3rd ACM International Conference on Recommender Systems*, pages 3–6. Citeseer, 2009.

[MA06] P. Massa and P. Avesani. Trust-aware bootstrapping of recommender systems. In *Proceedings of the ECAI 2006 Workshop on Recommender Systems*, pages 29–33, 2006.

[MA07a] P. Massa and P. Avesani. Trust-aware recommender systems. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 17–24, New York, New York, USA, 2007. ACM.

[MA07b] P. Massa and P. Avesani. Trust metrics on controversial users: balancing between tyranny of the majority and echo chambers. *International Journal on Semantic Web and Information Systems*, 3(1):39–64, 2007.

[MG11] Nicolas Marie and Fabien Gandon. Social Objects Description and Recommendation in Multidimensional Social Networks: OCSO Ontology and Semantic Spreading Activation. In *2011 IEEE Third Int'l Conference on Privacy,* *Security, Risk and Trust and 2011 IEEE Third Int'l Conference on Social Computing*, pages 1415–1420. IEEE, October 2011.

[MGML11] S. Meyffret, E. Guillot, L. Médini, and F. Laforest. RED: a Rich Epinions Dataset for Recommender Systems. Technical report, LIRIS UMR 5205 CNRS, INSA de Lyon, Lyon, 2011.

[MHN07] B. Mehta, T. Hofmann, and W. Nejdl. Robust collaborative filtering. In *Proceedings of the 2007 ACM conference on Recommender systems - RecSys '07*, pages 49–56, New York, USA, 2007. ACM Press.

[Mil67] S. Milgram. The small world problem. *Psychology today*, 2(1):60–67, 1967.

[MJ09] A. Montresor and M. Jelasity. PeerSim: A scalable P2P simulator. In *Proceeding of the 9th International Conference on Peer-to-Peer (P2P'09)*, pages 99–100. IEEE, September 2009.

[MKL09] H. Ma, I. King, and M.R. Lyu. Learning to recommend with social trust ensemble. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 203–210, New York, New York, USA, 2009. ACM.

[MML11] S. Meyffret, L. Médini, and F. Laforest. Trust-based recommendation with privacy. In *Inforsid2*, pages 369–384, 2011.

[MZL$^+$11] H. Ma, D. Zhou, C. Liu, M.R. Lyu, and I. King. Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 287–296, New York, New York, USA, 2011. ACM Press.

[OS05] J. O'Donovan and B. Smyth. Trust in recommender systems. In *Proceedings of the 10th international conference on Intelligent user interfaces*, pages 167–174, New York, USA, 2005. ACM.

[PB07] M. Pazzani and D. Billsus. Content-based recommendation systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The adaptive web*, pages 325–341. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[PBMW99] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web.

Technical report, Stanford Digital Library Technologies Project, 1999.

[PFW05] E. Pampalk, A. Flexer, and G. Widmer. Improvements of audio-based music similarity and genre classification. In *Crawford and Sandler*, volume 5, pages 628–633, 2005.

[PK09] G. Pitsilis and S.J. Knapskog. Social Trust as a solution to address sparsity-inherent problems of Recommender systems. *Recommender Systems & the Social Web*, 826(October):33–40, 2009.

[PT09] I. Pilászy and D. Tikk. Recommending new movies: even a few ratings are more valuable than metadata. *Proceedings of the third ACM conference on*, pages 93–100, 2009.

[QKSC11] D. Quercia, M. Kosinski, D. Stillwell, and J. Crowcroft. Our Twitter Profiles, Our Selves: Predicting Personality with Twitter. *2011 IEEE Third Int'l Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third Int'l Conference on Social Computing*, pages 180–185, October 2011.

[RD02] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 61–70, New York, New York, USA, 2002. ACM.

[RRSK11] F. Ricci, L. Rokach, B. Shapira, and P.B. Kantor, editors. *Recommender Systems Handbook*. Springer US, Boston, MA, 2011.

[SFHS07] J.B. Schafer, D. Frankowski, J.L. Herlocker, and S. Sen. Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer-Verlag, 2007.

[SKKR01] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In ACM, editor, *Proceedings of the tenth international conference on World Wide Web - WWW '01*, pages 285–295, New York, New York, USA, 2001. ACM Press.

[SKR99] J.B. Schafer, J. Konstan, and J. Riedi. Recommender systems in e-commerce. *Proceedings of the 1st ACM conference on Electronic commerce - EC '99*, pages 158–166, 1999.

[SM86] G. Salton and M.J. Mcgill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[VCC11] P. Victor, M. Cock, and C. Cornelis. Trust and Recommendations. In *Recommender Systems Handbook*, pages 645–675. Springer, 2011.

[WF94] S. Wasserman and K. Faust. *Social network analysis: Methods and applications*. Cambridge Univ Pr, 1994.

[Wil93] O.E. Williamson. Calculativeness, Trust, and Economic Organization. *Journal of Law and Economics*, 36(1):453–86, 1993.

[WL02] B. Whitman and S. Lawrence. Inferring descriptions and similarity for music from community metadata. In *Proceedings of the 2002 International Computer Music Conference*, pages 591–598, 2002.