Towards Scalable Personalization

THÈSE N^{O} 8299 (2018)

PRÉSENTÉE LE 23 FÉVRIER 2018
À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE DE PROGRAMMATION DISTRIBUÉE
PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Rhicheek PATRA

acceptée sur proposition du jury:

Prof. V. Kunčak, président du jury
Prof. R. Guerraoui, directeur de thèse
Prof. F. Taiani, rapporteur
Dr M. Bilenko, rapporteur
Prof. B. Faltings, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
Suisse
2018

TOWARDS SCALABLE PERSONALIZATION

RHICHEEK PATRA

The greatest challenge to any thinker is stating the problem in a way that will allow a solution.
— Bertrand Russell

In loving memory of my father…

Acknowledgements

To begin with, I am deeply thankful to my family, Tapasi Patra (my mother), Kusal Kumar Patra (my father) and Suvam Patra (my brother), for their endless support throughout my career. Pursuing a Ph.D. has been an overwhelming experience for me. Throughout my Ph.D., I had the wonderful opportunity to collaborate with amazing researchers, from both academia and industry, who made my experience even more amazing and these collaborations also subsequently led to this dissertation. To highlight these wonderful collaborations, I would use "we" instead of "I" throughout the main body of this dissertation.

First and foremost, I would like to thank my advisor, Rachid Guerraoui. Starting my Ph.D. right after my Bachelors, I did not have any significant experience in research. In this regard, Rachid was the ideal advisor for me due to his remarkable experience as an accomplished researcher. His constant support and encouragement throughout these years allowed me to grow as a researcher. I would also like to thank Anne-Marie Kermarrec, who co-supervised me on some projects during the initial two years of my Ph.D., for her insightful feedback and excellent discussions leading to high-quality research papers in top-tier conferences.

As I mentioned before, I had the wonderful opportunity to work with amazing researchers from industry throughout my Ph.D. duration. More precisely, I spent three summers as research intern at Technicolor (in 2015), Yandex (in 2016), and Oracle Labs (in 2017). These internships led to fruitful collaborations on diverse topics for which I am extremely thankful to my collaborators, namely, Erwan Le Merrer and Jean-Ronan Vigouroux (at Technicolor), Egor Samosvat, Michael Roizner and Andrei Mishchenko (at Yandex), and lastly Jinha Kim and Sungpack Hong (at Oracle Labs). These collaborations significantly improved my perspective on the practical impact of research in an industrial setting.

I also want to thank my excellent research colleagues from academia for our fruitful collaborations which led to multiple publications. During my first year, I had a very fruitful collaboration with Antoine Boutet and Davide Frey on scalability aspects of recommenders (Chapter 3), and also with Mahsa Taziki on users' privacy aspects (Chapter 5). The following years, I collaborated with Jingjing Wang towards another interesting work related to users' privacy (Chapter 5) as well as with Tao Lin on a project for designing recommenders for multi-application objective (Chapter 6). During the last year, I had another very fruitful collaboration with Georgios Damaskinos leading to interesting research papers on scalable recommenders (Chapter 4) and distributed machine learning.

Besides the above-mentioned collaborators, many thanks to my other amazing colleagues from the Distributed Programming Laboratory (LPD) as well as external collaborators for creating a great working environment. Mainly a huge thanks to Karolos Antoniadis, Vasileios Trigonakis, Tudor David, Davide Kozhaya, Georgios Chatzopoulos, Dragos-Adrian Seredinschi, El Mahdi El Mhamdi, Oana Balmau, Matej Pavlovic, Igor Zablotchi, Mahammad Valiyev, and Victor Bushkov. Also, a special thanks to Damien Hilloulin for the French version of the abstract of this thesis.

I would also like to express my special appreciation for the committee members of my PH.D. defense, namely, Boi Faltings, Viktor Kunčak, Misha Bilenko, and Francois Taiani. I want to thank EPFL, Google, and the European Research Council for financially supporting my research.

Being a researcher, I was more focused on the research aspects of academia. However, there are many administrative aspects associated with academics as well. In this regard, I want to thank the two secretaries of our lab, Kristine Verhamme and France Faille. I would also like to thank our system administrator Fabien Salvi for providing assistance during these years concerning any technical issues with LPD's computing resources.

Last but not the least, I also thank my close friends outside the lab who made the PH.D. years a lot more bearable. First, a big thanks to Monika Parmar for accompanying me throughout the last six years and most importantly for being there for me. Next, I also thank Saeid Sahraei and Hanjie Pan who started in the same year as me and with whom I got to (sometimes) go outside the lab.

EPFL, Lausanne, 18 January 2018 *Rhicheek Patra*

Preface

This dissertation presents the work that I did during my Ph.D. under the supervision of Professor Rachid Guerraoui at EPFL in Switzerland since September, 2013. This thesis focuses on three crucial aspects of personalization, namely, *Scalability* (Chapters 3 and 4), *Privacy* (Chapter 5), and *Heterogeneity* (Chapter 6). Throughout the duration of my Ph.D., I was involved in various research projects leading to high-quality research articles (mentioned below). The main results of this thesis appeared originally in the *highlighted* articles among the following. Moreover, the publications are ordered by the corresponding personalization aspects that they address.

Scalability

- Antoine Boutet, Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, and Rhicheek Patra (alphabetical order). *HyRec: leveraging browsers for scalable recommenders.* ACM/IFIP/USENIX Middleware, 2014.
- Georgios Damaskinos, Rachid Guerraoui, and Rhicheek Patra (alphabetical order). *Capturing the Moment: Lightweight Similarity Computations.* IEEE International Conference on Data Engineering (ICDE), 2017.
- Rachid Guerraoui, Erwan Le Merrer, Rhicheek Patra, and Jean-Ronan Vigouroux (alphabetical order). *Sequences, Items And Latent Links: Recommendation With Consumed Item Packs.* (Under submission, arXiv:1711.06100)
- Rachid Guerraoui, Erwan Le Merrer, Rhicheek Patra, and Bao-Duy Tran (alphabetical order). *Frugal Topology Construction for Stream Aggregation in the Cloud.* IEEE International Conference on Computer Communications (INFOCOM), 2016.
- Georgios Damaskinos, Rachid Guerraoui, and Rhicheek Patra (alphabetical order). *Mobile Learning: Distributed Machine Learning on Mobile Devices.* (Under submission)

Privacy

- Rachid Guerraoui, Anne-Marie Kermarrec, Rhicheek Patra, and Mahsa Taziki (alphabetical order). *D2P: Distance-Based Differential Privacy in Recommenders.* Proceedings of the 41st International Conference on Very Large Data Bases (PVLDB), 2015.
- Rachid Guerraoui, Anne-Marie Kermarrec, Rhicheek Patra, Mahammad Valiyev, and Jingjing Wang (alphabetical order). *I know nothing about you but here is what you might like.* 47th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2017.

HETEROGENEITY

- Rachid Guerraoui, Anne-Marie Kermarrec, Tao Lin, and Rhicheek Patra (alphabetical order). *Heterogeneous Recommendations: What You Might Like To Read After Watching Interstellar*. Proceedings of the 43rd International Conference on Very Large Data Bases (PVLDB), 2017.
- Rhicheek Patra, Egor Samosvat, Michael Roizner and Andrei Mishchenko. *BoostJet: Towards Combining Statistical Aggregates with Neural Embeddings for Recommendations*. (Under submission, arXiv:1711.05828)

EPFL, Lausanne, 18 January 2018 *Rhicheek Patra*

Abstract

The ever-growing amount of online information calls for *Personalization*. Among the various personalization systems, recommenders have become increasingly popular in recent years. Recommenders typically use *collaborative filtering* to suggest the most relevant items to their users.

The most prominent challenges underlying personalization are: *scalability*, *privacy*, and *heterogeneity*. *Scalability* is challenging given the growing rate of the Internet and its dynamics, both in terms of churn (i.e., users might leave/join at any time) and changes of user interests over time. *Privacy* is also a major concern as users might be reluctant to expose their profiles to unknown parties (e.g., other curious users), unless they have an incentive to significantly improve their navigation experience and sufficient guarantees about their privacy. *Heterogeneity* poses a major technical difficulty because, to be really meaningful, the profiles of users should be extracted from a number of their navigation activities (heterogeneity of *source* domains) and represented in a form that is general enough to be leveraged in the context of other applications (heterogeneity of *target* domains).

In this dissertation, we address the above-mentioned challenges. For scalability, we introduce *democratization* and *incrementality*. Our *democratization* approach focuses on *iteratively* offloading the computationally expensive tasks to the user devices (via browsers or applications). This approach achieves scalability by employing the devices of the users as additional resources and hence the throughput of the approach (i.e., number of updates per unit time) scales with the number of users. Our *incrementality* approach deals with incremental similarity metrics employing either explicit (e.g., ratings) or implicit (e.g., consumption sequences for users) feedback. This approach achieves scalability by reducing the time complexity of each update, and thereby enabling higher throughput.

We tackle the privacy concerns from two perspectives, i.e., anonymity from either other curious users (*user-level privacy*) or the service provider (*system-level privacy*). We strengthen the notion of *differential privacy* in the context of recommenders by introducing *distance-based differential privacy* (D2P) which prevents curious users from even guessing any category (e.g., genre) in which a user might be interested in. We also briefly introduce a recommender (X-REC) which employs *uniform user sampling* technique to achieve *user-level privacy* and an efficient homomorphic encryption scheme (X-HE) to achieve *system-level privacy*.

We also present a heterogeneous recommender (X-MAP) which employs a novel similarity metric (X-SIM) based on paths across heterogeneous items (i.e., items from different domains). To achieve a general form for any user profile, we generate her AlterEgo profile in a target domain by employing an item-to-item mapping from a source domain (e.g., movies) to a target domain (e.g., books). Moreover, X-MAP also enables differentially private AlterEgos. While X-MAP employs user-item interactions (e.g., ratings), we also explore the possibility of heterogeneous recommendation by using content-based features of users (e.g., demography, time-varying preferences) or items (e.g., popularity, price).

Keywords: personalization, recommender, machine learning, collaborative filtering, differential privacy, heterogeneity, similarity metric, scalability, energy efficiency, graph, distributed system.

Résumé

La quantité croissante d'informations en ligne appelle à la *Personnalisation*. Parmi les différents systèmes de personnalisation, les systèmes de recommandation sont devenus de plus en plus populaires ces dernières années. Les systèmes de recommandations utilisent généralement le *filtrage collaboratif* pour suggérer les éléments les plus pertinents à leurs utilisateurs.

Les défis les plus importants sous-jacents à la personnalisation sont : *l'évolutivité des systèmes* (« scalability«), *la confidentialité* et *l'hétérogénéité*. *L'évolutivité* est difficile compte tenu du taux croissant d'Internet et de sa dynamique, tant en termes de taux de désabonnement (c'est-à-dire, les utilisateurs peuvent quitter / rejoindre à tout moment) et les changements d'intérêts des utilisateurs au fil du temps. *La confidentialité* est également une préoccupation majeure car les utilisateurs peuvent être réticents à exposer leurs profils à des parties inconnues (par exemple, d'autres utilisateurs curieux), à moins d'être incités et d'améliorer significativement leur expérience de navigation et en garantissant leur confidentialité. *L'hétérogénéité* pose une difficulté technique majeure car, pour être vraiment significatif, les profils des utilisateurs doivent être extraits d'un certain nombre de leurs activités de navigation (hétérogénéité des domaines *sources*) et représenté sous une forme suffisamment générale pour être exploitée dans le contexte d'autres applications (hétérogénéité des domaines *cibles*).

Dans cette thèse, nous abordons les défis mentionnés ci-dessus. Pour l'évolutivité, nous introduisons *la démocratisation* et *l'incrémentalité*. Notre approche de *démocratisation* se concentre sur le transfert des tâches coûteuses en calcul vers les périphériques utilisateurs (via les navigateurs ou les applications) de manière *itérative*. Cette approche permet l'évolutivité en utilisant les dispositifs des utilisateurs en tant que ressources supplémentaires et par conséquent le débit de l'approche (c'est-à-dire le nombre de mises à jour par unité de temps) augmente avec le nombre d'utilisateurs. Notre approche *incrémentale* utilise des métriques de similarité incrémentale employant des retours explicites (par exemple, évaluations) ou implicites (par exemple, des séquences de consommation pour les utilisateurs). Cette approche permet une évolutivité en réduisant la complexité temporelle de chaque mise à jour et en permettant ainsi un débit plus élevé.

Nous abordons les problèmes de confidentialité sous deux angles, à savoir l'anonymat vis-à-vis des autres utilisateurs curieux (*confidentialité au niveau de l'utilisateur*) ou du fournisseur de services (*confidentialité au niveau du système*). Nous renforçons la notion de *confiden-*

tialité différentielle dans le contexte des systèmes de recommandation en introduisant la *confidentialité différentielle basée sur la distance* (« distance-based differential privacy« D2P) qui empêche les utilisateurs curieux de deviner ne serait-ce qu'une catégorie (par exemple, genre) dans laquelle un utilisateur pourrait être intéressé. Nous abordons aussi brièvement un système de recommandation (X-REC) qui utilise la technique *d'échantillonnage utilisateur uniforme* pour atteindre la *confidentialité au niveau de l'utilisateur* et un schéma de chiffrement homomorphique efficace (X-HE) pour atteindre la *confidentialité au niveau du système*.

Nous présentons également un système de recommandation hétérogène (X-MAP) qui utilise une nouvelle métrique de similarité (X-SIM) basée sur les chemins entre des éléments hétérogènes (c'est-à-dire, des éléments de différents domaines). Pour obtenir une forme générale pour n'importe quel profil utilisateur, nous générons son profil AlterEgo dans un domaine cible en utilisant un portage élément à élément d'un domaine source (par exemple des films) vers un domaine cible (par exemple, des livres). De plus, X-MAP permet également d'obtenir des profils AlterEgos privés au sens différentiel. Bien que X-MAP utilise des interactions utilisateur (par exemple des évaluations), nous explorons également la possibilité d'une recommandation hétérogène en utilisant les « caractéristiques de contenu » des utilisateurs (p. Ex., Démographie, préférences variables) ou des éléments (popularité, prix).

Mots-clés : personnalisation, recommandation, apprentissage automatique, filtrage collaboratif, confidentialité différentielle, hétérogénéité, métrique de similarité, évolutivité, efficacité énergétique, graphique, système distribué.

Contents

Acknowledgements **v**

Preface **vii**

Abstract (English) **ix**

Résumé (French) **xi**

List of Figures **xvii**

List of Tables **xxi**

1 Introduction **1**

- 1.1 Challenges in Personalization 1
  - 1.1.1 Scalability 1
  - 1.1.2 Privacy 2
  - 1.1.3 Heterogeneity 3
- 1.2 Contributions 3
  - 1.2.1 Scalability 3
  - 1.2.2 Privacy 5
  - 1.2.3 Heterogeneity 6
- 1.3 Roadmap 7

I Preliminaries **9**

2 Background **11**

- 2.1 Collaborative Filtering 11
  - 2.1.1 User-based collaborative filtering 12
  - 2.1.2 Item-based collaborative filtering 12
- 2.2 Temporal Relevance 13
- 2.3 Privacy 15

2.4 Heterogeneity 15
2.5 Gradient-descent Algorithm 16
2.6 Trust-distrust Relationship in Online Social Networks 17
2.7 Evaluation Metrics 17

II Scalability **21**

3 Democratization **23**
3.1 HyRec: Towards a hybrid architecture 23
    3.1.1 Overview 23
    3.1.2 HyRec 24
    3.1.3 Evaluation 27
    3.1.4 Related Work 33
    3.1.5 Conclusion 33
3.2 Extension to machine learning on mobile devices 34

4 Incrementality **39**
4.1 I-Sim: Incremental Similarity 39
    4.1.1 Overview 39
    4.1.2 I-Sim: A Novel Similarity 42
    4.1.3 I-Sim Applications 46
    4.1.4 Evaluation 50
    4.1.5 Related Work 58
    4.1.6 Conclusion 60
4.2 CIP: Consumed Item Packs 60
    4.2.1 Overview 60
    4.2.2 Consumed Item Packs (CIPs) 62
    4.2.3 CIP Algorithms 63
    4.2.4 Implementation 68
    4.2.5 Evaluation 70
    4.2.6 Related Work 74
    4.2.7 Conclusion 76

III Privacy **77**

5 Privacy **79**
5.1 User-level Privacy 79
    5.1.1 Overview 79

5.1.2 D2P: Privacy for Recommenders . . . 81
5.1.3 D2P-based Recommender . . . 83
5.1.4 Evaluation . . . 88
5.1.5 Related Work . . . 95
5.1.6 Conclusion . . . 97
5.2 System-level Privacy . . . 98

IV Heterogeneity 99

6 Heterogeneous Recommendations 101
6.1 Heterogeneous Recommendations with Alter-Egos . . . 101
6.1.1 Overview . . . 101
6.1.2 Heterogeneous Recommendation Problem . . . 103
6.1.3 X-SIM: Cross-domain similarity . . . 104
6.1.4 X-MAP: Cross-domain recommender . . . 107
6.1.5 Implementation . . . 114
6.1.6 Evaluation . . . 116
6.1.7 Related Work . . . 124
6.1.8 Conclusion . . . 125
6.2 Content-enabled Heterogeneous Recommendations . . . 125

V Thesis Conclusions and Remarks 127

7 Concluding Remarks 129
7.1 Summary and Implications . . . 129
7.2 Future Work . . . 130

VI Appendices 133

8 Appendices 135
8.1 Appendix A: I-SIM . . . 135
8.1.1 Correctness proofs . . . 135
8.2 Appendix B: X-MAP . . . 140
8.2.1 Correctness proofs . . . 140
8.2.2 Additional experiments . . . 142

Bibliography 145

Curriculum Vitae 161

List of Figures

3.1 Centralized, decentralized and hybrid (HYREC) architecture of a recommender. 25
3.2 Timeline: a centralized approach vs. HYREC 26
3.3 Average view similarity on ML1 dataset for HYREC and ideal KNN. 29
3.4 Convergence of the candidate set size (ML1 dataset). 30
3.5 Recommendation quality on the ML1 dataset for HYREC as well as offline and online ideal KNN ($k = 10$). 31
3.6 Time to compute the k nearest neighbors on ML and Digg workloads. 32
3.7 The architecture overview of HYML. 35
3.8 Throughput comparison among single nodes (mobile, CPU, GPU) and a cluster. 37

4.1 Temporal effects in ML-1M dataset. 40
4.2 Limitations of state-of-the-art similarity metrics with respect to temporal relevance and incremental updates. The gray areas in the right subfigure indicate the similarities (S_{ij}) that need to be updated within a time interval $[t-\delta, t]$. . . 41
4.3 Comparison between incremental (I-SIM) and non-incremental similarity computations [157, 5] for ML-1M dataset. The time interval for the active users is 5 days. 44
4.4 The architecture overview of SWIFT. 46
4.5 The biased sampling technique of SWIFT. 48
4.6 Voters' trust in an administrator during a Wiki-Election 50
4.7 Impact of model size (K) and recommendations-per-click (N) on accuracy. . . 53
4.8 Impact of flexibility mode on accuracy for ML-1M. 54
4.9 Impact of temporal relevance (α) on accuracy. Setting α to 0 deactivates SWIFT's temporal feature. 55
4.10 Impact of model size (K) on latency (stream processing). 56
4.11 Impact of batch processing on latency for ML-1M. 56
4.12 Scalability comparison for ML-20M. 57
4.13 Impact of item popularity on energy consumption for ML-20M. 58

4.14 Existence of temporal consumption habits of users in MovieLens dataset. . . . 61
4.15 Topology and data structures for CIP-U and CIP-I (arrows denote the RDD dependencies). 69
4.16 Topology and data structures for DEEPCIP. 70
4.17 Recommendation quality of CIP-based algorithms versus competitors. 72
4.18 Partition effects. 74
4.19 Cluster size effects. 75

5.1 D2P Illustration. 80
5.2 D2P-based Recommender. 84
5.3 D2P Modification Phase. 85
5.4 ML1 Dataset Partitions based on rating density. 89
5.5 Recall@N with varying Dataset Characteristics. 90
5.6 Effect of Profiler Probability (p^*) on Quality for the ML Dataset (User-based CF). 91
5.7 Effect of Profiler Probability (p^*) on Quality for the Jester Dataset (User-based CF). 91
5.8 Effect of Selector Probability (p) on Quality for the ML Dataset (User-based CF). 92
5.9 Effect of Selector Probability (p) on Quality for the Jester Dataset (User-based CF). 92
5.10 Effect of Distance Metric (λ) on Quality for the ML Dataset (User-based CF). . . 92
5.11 Distance for Personal Choice. 93
5.12 Distance for Average. 93
5.13 Privacy Parameters Comparison. 94
5.14 Catalog Coverage Comparison. 95
5.15 Effect of Selector Probability (p) on Quality for the ML Dataset (Item-based CF). 96
5.16 Effect of Profiler Probability (p^*) on Quality for the ML Dataset (Item-based CF). 96

6.1 Heterogeneous recommendation using meta-paths. 101
6.2 Layer-based pruning in X-MAP. 105
6.3 Alice's AlterEgo profile (in target domain) mapped from her original profile (in source domain). 108
6.4 The components of X-MAP: *Baseliner, Extender, Generator, Recommender*. . . . 114
6.5 Temporal relevance (X-MAP, NX-MAP). 119
6.6 Privacy-quality trade-off in X-MAP-IB. 119
6.7 Privacy-quality trade-off in X-MAP-UB. 120
6.8 MAE comparison with varying k. 120
6.9 MAE comparison (Overlap size).[1] 121
6.10 MAE comparison based on profile size. 121
6.11 Scalability of X-MAP. 123

6.12 Feedback from 51 users over 1 week. 123

List of Tables

2.1 Confusion Matrix for true/false positive/negative recommendations. 18

3.1 Datasets statistics 28
3.2 Impact on the cost of a centralized back-end server according to the KNN selection period. 33

4.1 Model comparison (MAE) between incremental and non-incremental alternatives. 55
4.2 Runtime and accuracy comparisons for I-TRUST and C-TRUST. 58
4.3 Details of the datasets used in our experiments. 71

5.1 Datasets characteristics. 90
5.2 Overhead of Privacy. 95

6.1 Sub-domains (D_1 and D_2) based on genres in Movielens 20M dataset. 122
6.2 MAE comparison (homogeneous setting on ML-20M dataset). 122

8.1 Comparison between user-based (UB) and item-based (IB) recommenders in different deployment scenarios with Amazon datasets. Bold denotes faster computation time relative to the alternative. 143
8.2 MAE comparison between NX-MAP, X-MAP and MLLIB-ALS on Amazon datasets.143
8.3 Densities for two domains in the Amazon dataset. 144

1 Introduction

In the modern generation of web-based services, the number of users is increasing continuously. This number bumped up from 16 million users in 1995 to 3.6 billion users in 2017. Such an immense growth in the number of users evidently led to an exponential increase in the amount of data available online (about 2.5 billion GB of data are created everyday). As a result, the web has become a big storehouse of information, making it impossible for any individual to explore the whole web contents to extract relevant data. Subsequently, personalization [25] has become an essential tool to navigate this wealth of information available on the Internet. Particularly popular now are recommender systems [107] which provide users with personalized content, based on their past online behavior (e.g., browsing history, clicks) and that of other similar users. These systems have been successfully employed by major online retailers such as Amazon to propose new items to their customers. Social networking sites, such as Facebook or Twitter, exploit these systems to suggest friends/followers to their users and to filter the content displayed on their feeds. Google or Yahoo! use these systems to provide personalized news to registered users. Personalization has now become ubiquitous in social media platforms and employed by almost all big players (e.g., Google, Facebook, Amazon) as well as relatively smaller ones (e.g., startups).

1.1 Challenges in Personalization

While appealing, building such personalization systems raises several technical challenges. We discuss about these technical challenges in the following.

1.1.1 Scalability

As we mentioned earlier that the growth in the number of online users led to the emergence of personalization. However, the personalization schemes also need to be *scalable* in order to process the ever-growing amount of information created by the users. Personalization schemes, employed to *build* recommender systems, demand immense amount of computing resources to process this huge volume of information and provide relevant personalized content. Moreover, any such recommender system continuously needs to be updated due to an ever-increasing amount of data, collected from online platforms, with ever-changing patterns due to various factors e.g., popularity of items or behavioral trends of users [45, 193]. Billion-dollar companies such as Google or Facebook leverage their personal huge data centers to distribute the computations for updating the recommender system using the incoming data either in an *online* manner (e.g.., stream processing [92]) or in an *offline* manner (e.g., batch processing [163] at predetermined periodic intervals). For relatively smaller service providers, the most practical option is to employ cloud-based computing resources such as Amazon EC2 or Microsoft Azure but only in an offline manner. As a result, there are significant investments (i.e., Total Cost of Ownership) involved in employing such cloud-based resources. Dimension reduction and algorithmic optimizations [74, 66], or sample-based approaches [54, 61, 51], partially tackle the problem by reducing the time complexity of each update at the cost of performing more updates in a parallelized fashion. Yet there exists the need of significant investments in computational resources with growing number of users and items [143, 46, 28]. Even with massive parallelization (map-reduce [49]) on multicore architectures [151, 130] or elastic cloud architectures [46], personalization remains expensive in terms of both hardware and energy consumption [38, 133]. In this thesis, the main technical challenge concerning scalability is to design novel solutions that significantly reduce the number of computations (i.e., time complexity) for updating the recommender system and thereby also reduce the investment in computing resources.

1.1.2 Privacy

The growing tendency towards personalization has raised several privacy concerns [150] as more and more personal data is being collected and used by various personalization services. It is often observed that when an Internet user accesses some service, the provider of this service typically claims the ownership of any personal information provided by the user. The service provider sometimes even distributes the collected information to third parties like advertising and promotional partners [168]. Even the sharing of anonymized user information like the Netflix Prize dataset might end up not being secure. For instance, Narayanan et. al presented a de-anonymization attack that linked the records in the Netflix Prize dataset with the IMDb profiles available publicly [139].

Personalization systems like recommenders are particularly fragile with respect to privacy due to their ability to provide *serendipitous* recommendations (i.e., *unexpected* but *desired* recommendations) [150]. Recommender systems typically make predictions about the preferences of any user by analyzing the preferences of other users. Hence, recommenders are particularly vulnerable to privacy attacks as they directly rely on information about users to provide relevant recommendations. Recommenders aggregate *user preferences* [152] in ways analogous to database queries, which can be exploited by adversaries to extract personal identifiable information about a specific user [150]. In this thesis, the primary challenge concerning privacy is to provide novel privacy-preserving solutions (with some formal guarantees) which do not affect the recommendation quality significantly, and also do not require a significant computation overhead for the privacy preservation.

1.1.3 Heterogeneity

Although widely used today, recommender systems are mainly applied in a *homogeneous* sense: movie recommenders like IMDb or Netflix, news recommenders like Google News or Yahoo News, as well as music recommenders like Last.fm or Spotify, each focuses on a single specific application domain. In short, you will be recommended books only if you rated books, and you will be recommended movies only if you rated movies. Given the growing multiplicity of web applications, homogeneity is a major limitation. For example, with most state-of-the-art recommenders, Alice who just joined a book-oriented web application, and never rated any book before, cannot be recommended any relevant book, even if she has been rating many movies. This example is a classical illustration of the *cold-start* problem [159] in recommender systems.

Heterogeneous preferences on the web, i.e., preferences from multiple application domains, could be leveraged to improve personalization furthermore, not only for users who are new to a domain (i.e., cold-start situation), but also when the data is *sparse* [2] (e.g., a very few ratings per user). The scalability and privacy challenges become even more crucial in heterogeneous recommenders due to increasing connections across users and items from multiple domains or applications. In this thesis, the technical challenge is to design a private and scalable heterogeneous recommender which provides relevant recommendations across multiple domains or applications.

1.2 Contributions

In this thesis, we address the above-mentioned technical challenges concerning personalization. We present the main contributions of this thesis in the following which are ordered by the topics.

1.2.1 Scalability

In this thesis, we focus on two primary directions that improves the scalability of current state-of-the-art personalization systems. We improve scalability by designing *iterative* or *incremental* solutions that significantly reduce the number of computations for updating the recommender system.

A. Democratization

In our first step, we focus on the *democratization* of computationally expensive jobs for updating the recommender. We use the notion of *democratization* since our solution can be easily deployed by any service provider irrespective of the available computational resources. The objective here is to offload *customized* computation jobs to computational devices which could be high-end devices like laptops as well as lightweight devices like smartphones or tablets. Typically, these are the devices of the end-users who are using the personalization service for getting relevant suggestions. The computation jobs are customized based on the computational capacity of the corresponding computing device. In Chapter 3.1 of this thesis, we introduce HYREC [27] which offloads computational jobs of constant time complexity to the devices. The motivation of this work is to explore solutions that can *democratize personalization* by making it accessible to any content-provider company, without requiring huge investments. HYREC employs an *iterative* technique to update the nearest-neighbor graph [174] of users. Such an iterative solution also scales out with an increasing number of users as the throughput, in terms of the number of updates, increases with more devices from the users. HYREC employs a *hybrid* architecture capable of providing a cost-effective personalization platform to web-content services. Instead of scaling through either larger and larger recommendation back-end servers, or through fully decentralizing the recommendation process by relying solely on the front-end clients, HYREC delegates expensive computation tasks to the clients while, at the same time, retaining on the server side the recommender's coordination tasks and the maintenance of the user-user graph (i.e., nearest-neighbor graph) which reflects the relationship between different users.

We also give a brief overview regarding how we can extend this idea of iteratively offloading computational jobs to Machine Learning (ML) applications. In Chapter 3.2 of this thesis, we present HYML, an extension of HYREC, which offloads computational jobs proportional to the device features (e.g., available memory, cpu cores). Unlike collaborative filtering employed in HYREC, performance variability, due to varying device features, poses a significant challenge [196] to train any *centralized* ML algorithm (i.e., a global ML model stored and updated on a central server) by employing users' devices. Due to the asynchronous nature [120] of the training procedure (i.e., model updates) combined with the heterogeneity of the mobile devices, there exists significant performance difference between the slow and fast mobile devices. HYML currently employs classical heterogeneity-aware model update algorithms [97, 196] in such a heterogeneous environment of mobile computing devices.

B. Incrementality

We next focus on the *incrementality* of the updates for the recommender system. At the heart of many practical collaborative filtering techniques [92] lies the computation of *similarities* between users, also known as *like-mindedness.* Even for trust-distrust prediction in Online Social Networks (OSNs), nearest neighbor graphs employ similarities between the nodes [197]. We observe that existing similarity metrics [157, 187] were not designed to handle a very large number of users with rapidly changing behavior. Moreover, recommenders typically collect user *preferences* using *explicit* feedback such as *numerical* ratings (star ratings in IMDb, Netflix, Amazon), *binary* preferences (likes/dislikes in Youtube), or *unary* preferences (retweets in Twitter). However, in systems where the item catalog is large, users tend to give explicit feedback on a trace amount of those items leading the classical *sparsity* issue [2]. This led to the usage of recommenders employing implicit feedback (e.g., time corresponding to purchase events [115] or purchase sequences [39]). We provide incremental solutions for recommender systems employing either explicit feedback (in Chapter 4.1) or implicit feedback (in Chapter 4.2).

In Chapter 4.1 of this thesis, we introduce a novel similarity metric, we call I-SIM [45], which enables lightweight similarity computations incorporating the rapidly changing temporal behavior of users. I-SIM can be considered as a "temporalization" of the adjusted cosine similarity [157] and hence of the cosine similarity which is a specific instance of adjusted cosine similarity. Therefore, I-SIM can be easily integrated with other time-aware applications in Online Social Networks (OSNs) e.g., trust-distrust predictions. I-SIM is *lightweight* in the sense that it can be updated *incrementally* to achieve low latency and limited energy consumption. Lastly, we highlight the fact that I-SIM employs explicit feedback from users (e.g. ratings) for the incremental updates.

As we mentioned above, relying on explicit feedback raises issues regarding feedback *sparsity* (thereby impacting the quality of recommendations [2]), and limited efficiency for recommending fresh items in reaction to recent user actions [122]. We investigate the existence of a higher level abstraction for sequences of consumed items, and algorithms for dealing with them. In Chapter 4.2 of this thesis, we introduce the notion of *consumed item packs* (CIPS [78]) to extract relevant implicit information from consumption history logs of users. We propose novel algorithms using CIPS. To address scalability, the CIP-based algorithms are *incremental*: they enable to incorporate fresh items consumed recently by users, in order to update the recommendations in an efficient manner.

1.2.2 Privacy

In this thesis, we investigate how we can protect the privacy of users while providing personalized recommendations. We consider two levels of privacy. The first level is to protect the privacy of any user from other curious users (who can perform attacks [31]) which we denote as *user-level privacy*. The second level is to protect the privacy of users from the service provider itself which we denote as *system-level privacy*.

In Chapter 5.1 of this thesis, we present D2P, a novel protocol that uses a probabilistic substitution technique to create the AlterEgo profile of an original user profile. D2P ensures a strong form of *differential privacy* [55, 57], which we call **D***istance-based* **D***ifferential* **P***rivacy* [76]. Differential privacy [55, 57] is a celebrated property, originally introduced in the context of databases. Intuitively, it ensures that the removal of a record from a database does not change the result of a query to that database - modulo some arbitrarily small value (ϵ). In this sense, the presence in the database of every single record - possibly revealing some information about some user - is anonymous as no query can reveal the very existence of that record to any other user (modulo ϵ). Applying this notion in the context of recommenders would mean that - modulo ϵ - no user v would be able to guess - based on the recommendations she gets - whether some other user u has some item i in her profile, e.g., whether u has seen some movie i. Such a guarantee, however, might be considered weak as nothing would prevent v from guessing that u has in her profile some item that is very similar to i, e.g., that u has seen some movie similar to i. We strengthen the notion of differential privacy in the context of CF recommenders to guarantee that any user v is not only prevented from guessing whether the profile of u contains some item i, but also whether the profile of u contains any item i' within some *distance* λ from i (say any movie of the same category of i): hence the name **D***istance-based* **D***ifferential* **P***rivacy* (D2P). Our D2P protocol ensures this property.

In Chapter 5.2 of this thesis, we provide a brief overview of how we design X-REC [77], a novel recommender which ensures the privacy of users against the service provider (*system-level privacy*) or other curious users (*user-level privacy*) with negligible increase of latency in providing recommendations to end-users (due to the privacy overhead), while preserving recommendation quality. X-REC employs a *uniform user sampling* technique to achieve *user-level privacy* and an efficient homomorphic encryption scheme (X-HE) to achieve *system-level privacy*.

1.2.3 Heterogeneity

In Chapter 6 of this thesis, we present a heterogeneous recommender which we call X-MAP: *Cross-domain personalization system* [75]. X-MAP employs a novel similarity metric, X-SIM, which computes a meta-path-based[1] transitive closure of inter-item similarities across several domains. X-SIM involves adaptations, to the heterogeneous case, of classical *significance weighting* [84] (to account for the number of users involved in a meta-path) and *path length* [150] (to capture the effect of meta-path lengths) schemes. X-MAP also employs the notion of *AlterEgos*, namely artificial profiles (created using X-SIM), of users even in domains where they have *no* or *very little* activity yet. We generate an AlterEgo profile (of Alice) in a target domain leveraging an item-to-item mapping from a source domain (e.g., movies) to the target domain (e.g., books). AlterEgos enable to integrate any standard recommendation feature in the target domain and preserve, for example, the temporal behavior of users [53] across the domains. X-MAP also provides differential privacy by using an obfuscation mechanism, based on the meta-path-based similarities, to guarantee differentially private AlterEgos. We also briefly explore the possibility to perform content-enabled heterogeneous recommendations [144] by employing statistical aggregates of user features (e.g., demography, time-varying preferences) or item features (e.g., popularity, price).

---

[1] A meta-path in a heterogeneous graph G can be defined as a sequence of adjacent heterogeneous vertices (e.g., movies or books) connected by edges in G.

1.3 Roadmap

The rest of this thesis is organized as follows.

PART I

- Chapter 2 presents some preliminary concepts in recommender systems, namely, collaborative filtering, differential privacy, and temporal relevance, along with the standard quality metrics used to evaluate the recommenders.

PART II

- Chapter 3 presents the democratization approach for recommender systems (HyRec) and then provides a brief overview regarding how the idea can be extended to classical machine learning applications (HyML).
- Chapter 4 presents two incrementality approaches for scalability depending on the type of the feedback which could be either *explicit* (I-Sim) or *implicit* (CIP).

PART III

- Chapter 5 presents our notion of distance-based differential privacy (D2P) which strengthens the notion of classical differential privacy used for providing user-level privacy in recommenders. We also provide a brief overview regarding how we can achieve system-level privacy besides user-level privacy by employing X-Rec.

PART IV

- Chapter 6 presents a heterogeneous recommender system (X-Map) which enables recommendations across multiple domains based on user-item interactions (e.g., ratings). We also briefly explore content-enabled heterogeneous recommendations.

PART V

- Chapter 7 summarizes the contributions of this thesis along with its implications at a high level. We also highlight some interesting research directions as potential future work that the contributions of this thesis enable.

PART VI

- Chapter 8 provides some supplementary materials (e.g., correctness proofs, additional experiments) for interested readers.

PART I

Preliminaries

In this part of the thesis, we present the primary background concepts required for understanding the various personalization-related approaches covered in this thesis. These concepts are elementary to this thesis and we refer to them throughout the rest of the chapters.

2 Background

We recall here the classical notions of collaborative filtering, temporal relevance, heterogeneity, and privacy. Other than these standard concepts, we also provide a brief overview of trust-distrust predictions in online social networks, the classical gradient-descent algorithm, and the standard metrics for evaluating recommenders.

2.1 Collaborative Filtering

Collaborative Filtering (CF) algorithms fall mainly in two categories: *memory-based* [154, 170] and *model-based* [89, 134, 180]. Memory-based algorithms typically compute the *top-k* like-minded users for any given user (say Alice), denoted as the *neighbors* of Alice, from the training database, and then make recommendations to Alice based on the rating history of her neighbors. In contrast to memory-based algorithms, model-based ones first extract some information (also known as *features*) about users (including Alice) from the database to train a *model* and then use this model to make recommendations for the users (including Alice). Memory-based algorithms are more flexible in practice compared to model-based ones [92]. It is relatively more time-consuming to add new incoming data to model-based systems because training a model takes significant amount of time depending on the complexity of the model along with the hyper-parameter tuning.

Neighbor-based CF, based on *k nearest neighbor* (KNN) algorithms, are very popular and widely used in practice [157, 83]. The goal is to find similar objects (users or items) by exploring the relationships between them. The primary techniques employed by recommenders to explore these relationships can be divided into two categories: *user-based* and *item-based.* A user-based technique predicts a target user's preference for an item by leveraging the rating information aggregated from similar users. An item-based technique applies the same approach, but utilizes similarities between items instead of users.

We now provide a detailed explanation of the user-based and item-based collaborative filtering. We start with presenting the recommendation setting. We consider a database consisting of

$\mathcal{N}$ rating events on a set of m items $\mathcal{I} = \{i_1, i_2, .., i_m\}$ by a set of n users $\mathcal{U} = \{u_1, u_2, ..., u_n\}$ over time. The ratings are sorted based on the time of the event. Each rating event is in the form of a tuple: $\langle u, i, r_{ui}, \tau_{ui} \rangle$ which reflects the fact that user u provided a rating r_{ui} for an item i at a timestamp τ_{ui}. Furthermore, $\mathcal{U}_i^t$ denotes the set of users who have rated i until timestep t.

2.1.1 User-based collaborative filtering

A user-based CF scheme typically consists of three phases as shown in Algorithm 1. We describe each of these phases in the following.

Similarity computation phase. This phase concerns with the similarity computations based on the observed ratings. We use the pearson correlation or cosine similarity [60] as the similarity metric for this phase.

Neighborhood computation phase. This phase deals with computing the most similar users corresponding to a given user, based on the computed similarities from the previous phase, and then creating the user-user network. For each user u, the top-K users, i.e., with the K highest similarities, are selected as the neighbors. The parameter K denotes the *model size.*

Prediction phase. In this phase, there are either *prediction*-based approach which predict the scores for every item (or a filtered set of items) typically according to Equation 2.2 or *popularity*-based approach where the recommendations are the most popular items from a given user's neighborhood.

2.1.2 Item-based collaborative filtering

A standard item-based CF scheme typically consists of three phases as shown in Algorithm 2. We briefly describe each of these phases in the following.

Similarity computation phase. This phase concerns with the similarity computations based on the observed ratings. We mostly use the adjusted cosine similarity as it was empirically demonstrated to be superior to other metrics for item-based CF [157]. The deviation from the average rating effectively captures the user's rating behavior. Moreover, the ratings provided by users that generally give low (strict) or high (generous) ratings, have a uniform effect on the similarities.

Neighborhood computation phase. This phase deals with computing the most similar items corresponding to any given item, based on the computed similarities, and creating the item-item network. For each item i, the top-K items, i.e., with the K highest similarities, are selected as the neighbors. The parameter K denotes the *model size.*

Prediction phase. In this phase, the prediction scores are computed for each item according to Equation 2.4. Note that subtracting a user's average rating $\bar{r_u}$ compensates for differences in her rating scale thus making predictions even more accurate.

Algorithm 1 Standard User-based CF

```
Require: $\mathcal{I}$: Item set; $\mathcal{U}$: User set; $\mathcal{I}_u$: Set of Items rated by a user with user-id $u$.
Ensure: $R_a$: Top-$N$ recommendations for a user Alice ($a$)

Phase 1 - Similarity computation: GETSIMILARS($a$,$\mathcal{U}$)

Ensure: $s_a$: Dictionary for user $a$ with user-ids as keys and similarities as values.
for $u$ in $\mathcal{U}$ do
    $s_a[u] = \frac{\sum_{i \in \mathcal{I}_u \cap \mathcal{I}_a} (r_{ui} - \bar{r_u})(r_{ai} - \bar{r_a})}{\sqrt{\sum_{i \in \mathcal{I}_u} (r_{ui} - \bar{r_u})^2} \sqrt{\sum_{i \in \mathcal{I}_a} (r_{ai} - \bar{r_a})^2}}$    (2.1)
end for
return: $s_a$

Phase 2 - Neighborhood computation: kNN ($a$,$\mathcal{U}$)

Ensure: $N_a$: $K$ most similar users to user $a$.
$N_a$ = NLARGEST($K$, GETSIMILARS($a$,$\mathcal{U}$))
return: $N_a$

Phase 3 - Prediction: TOPN($\mathcal{U}$)

Require: $S_{av}$: similarity between two users $a$, $v$.
Ensure: $R_a$: Top-$N$ recommendations for Alice.
var PRED                                        ▷ Dictionary with predictions for Alice
for $i$ in $\mathcal{I}$ do
    $\text{PRED}[i] = \bar{r_a} + \frac{\sum_{v \in \text{kNN}(a,\mathcal{U}) \cap \mathcal{U}_i} (r_{vi} - \bar{r_v}) S_{av}}{\sum_{v \in \text{kNN}(a,\mathcal{U}) \cap \mathcal{U}_i} |S_{uv}|}$    (2.2)
end for
$R_a$ = NLARGEST($N$, PRED)
return: $R_a$
```

2.2 Temporal Relevance

Temporal relevance [110, 122] is a popular notion in data mining, commonly known as *concept drift*, a dynamic learning problem over time. A typical example is the change in user's interests when following an online news stream. In such domains (e.g. news, deals), the target concept (user's interests) depends on some temporal context (e.g., mood, financial state). This constantly changing context can induce changes in the target concepts, producing a concept drift. We now provide the definition of temporal relevance at any given timestep as follows where

```
Algorithm 2 Standard Item-based CF
Require: $\mathcal{I}$: Item set; $\mathcal{U}$: User set; $\mathcal{U}_j$: Set of users who rated an item with item-id $j$; $\bar{r_u}$: Average rating for user $u$.
Ensure: $R_a$: Top-$N$ recommendations for a user Alice ($a$)

Phase 1 - Similarity computation: GetSimilars($j$,$\mathcal{I}$)

Ensure: $s_j$: Dictionary for item $j$ with item-ids as keys and similarities as values.
for $i$ in $\mathcal{I}$ do
    $$s_j[i] = \frac{\sum_{u \in \mathcal{U}_i \cap \mathcal{U}_j} (r_{ui} - \bar{r_u})(r_{uj} - \bar{r_u})}{\sqrt{\sum_{u \in \mathcal{U}_i} (r_{ui} - \bar{r_u})^2} \sqrt{\sum_{u \in \mathcal{U}_j} (r_{uj} - \bar{r_u})^2}} \qquad (2.3)$$
end for
return: $s_j$

Phase 2 - Neighborhood computation: kNN ($j$,$\mathcal{I}$)

Ensure: $N_j$: $K$ most similar items to item $j$.
$N_j$ = nLargest($K$, GetSimilars($j$,$\mathcal{I}$))
return: $N_j$

Phase 3 - Prediction: TopN($\mathcal{I}$)

Require: $S_{ij}$: similarity between two items $i$, $j$.
Ensure: $R_a$: Top-$N$ recommendations for Alice.
var Pred                                        ▷ Dictionary with predictions for Alice
for $i$ in $\mathcal{I}$ do
    $$\text{Pred}[i] = \bar{r_a} + \frac{\sum_{j \in \text{kNN}(i,\mathcal{I}) \cap \mathcal{I}_a} (r_{aj} - \bar{r_a}) S_{ij}}{\sum_{j \in \text{kNN}(i,\mathcal{I}) \cap \mathcal{I}_a} |S_{ij}|} \qquad (2.4)$$
end for
$R_a$ = nLargest($N$, Pred)
return: $R_a$
```

timestep is a logical time corresponding to the current number of incremental updates.

Definition 1 (Temporal Relevance). *Temporal relevance measures the relevance of a feedback* s_{ui} *for making predictions at a timestep* t *based on a time-decaying parameter* α. *In the following, we denote the temporal relevance of* s_{ui} *at a timestep* t *by* $f_{ui}^{\alpha}(t)$ *and assign a weight* *to s_{ui} depending on the interval since the timestep (t_{ui}) when the actual feedback was provided.*

$$f_{ui}^{\alpha}(t) = e^{-\alpha(t-t_{ui})} \tag{2.5}$$

Temporal relevance can be incrementally updated as follows: $f_{ui}^{\alpha}(t+1) = e^{-\alpha} f_{ui}^{\alpha}(t)$. We consider one decay factor (Equation 2.5). However, multiple weighting factors like temporal regression [29] based ones might also be considered.

2.3 Privacy

Privacy is another crucial aspect in recommender systems and preserving privacy in CF recommenders is challenging. It was shown using the Netflix Prize dataset that even anonymizing individual data before releasing it publicly is not enough to preserve privacy [139]. Even cryptographic approaches do not preclude the possibility of the output leaking information about the personal input of individuals [181]. The need for stronger and robust privacy guarantees motivated the emergence of the notion of *Differential Privacy* [55, 57, 64]. First introduced in the context of databases, differential privacy provides *quantifiable* privacy guarantees.

Differential Privacy

Differential privacy [58] was initially devised in a context where statistical information about a database is released without revealing information about its individual entries. Differential privacy provides formal privacy guarantees that do not depend on an adversary's background knowledge (including access to other databases) or computational power. More specifically, differential privacy is defined as follows.

Definition 2 (DIFFERENTIAL PRIVACY [58])**.** *A randomized function $\mathcal{R}$ ensures ϵ-differential privacy if for all datasets D_1 and D_2, differing on at most one user profile, and all $t \in Range(\mathcal{R})$, the following inequality always holds:*

$$\frac{Pr[\mathcal{R}(D_1) = t]}{Pr[\mathcal{R}(D_2) = t]} \leq exp(\epsilon) \tag{2.6}$$

Remark 1 (COMPOSITION IN DIFFERENTIAL PRIVACY [59])**.** *Let $\mathcal{R}_1$ be an ϵ_1-differentially private algorithm, and $\mathcal{R}_2$ be an ϵ_2-differentially private algorithm. Then, their composition, i.e., $\mathcal{R}_{1,2}(x) = (\mathcal{R}_1(x), \mathcal{R}_2(x))$, is $\epsilon_1 + \epsilon_2$-differentially private.*

2.4 Heterogeneity

The multiplicity of web domains (movies, books, songs) is calling for *heterogeneous* recommenders that could utilize ratings for one domain (i.e., *source*) to provide recommendations in another one (i.e., *target*). Without loss of generality, we formulate the heterogeneity problem using two domains, referred to as the source domain ($\mathcal{D}^S$) and the target domain ($\mathcal{D}^T$). We note that the problem of heterogeneous recommendations trivially extends to multiple source and multiple target domains. We use superscript notations ^{S} and ^{T} to differentiate the source and the target domains. We assume that users in $\mathcal{U}^S$ and $\mathcal{U}^T$ overlap[1], but $\mathcal{I}^S$ and $\mathcal{I}^T$ have no common items. This captures the most common heterogeneous personalization scenario in e-commerce companies such as Amazon or eBay nowadays. The heterogeneous recommendation objective is to recommend items in $\mathcal{I}^T$ to users in $\mathcal{U}^S$ based on the preferences of $\mathcal{U}^S$ for $\mathcal{I}^S$ (ratings in the source domain), $\mathcal{U}^T$ for $\mathcal{I}^T$ (ratings in the target domain) and $\mathcal{U}^S \cap \mathcal{U}^T$ for $\mathcal{I}^S \cup \mathcal{I}^T$ (overlapping ratings across the domains). In other words, we aim to recommend items in $\mathcal{I}^T$ to a user who rated a few items (sparsity) or no items (cold-start) in $\mathcal{I}^T$. For instance, we intend to recommend new relevant books (i.e., items in $\mathcal{D}^T$) either to Alice who never rated any book (cold-start) or to Bob who rated only a single book (sparsity). Both the users (Alice and Bob) rated a few movies (i.e., items in $\mathcal{D}^S$).

2.5 Gradient-descent Algorithm

Machine learning approaches typically focus on solving the following optimization problem [120].

$$\min_{\boldsymbol{\theta} \in \mathcal{R}^d} f(\boldsymbol{\theta}) := E_{\xi}[F(\boldsymbol{\theta}; \xi)]$$

where $\xi \in \Xi$ is a random variable and $f(\boldsymbol{\theta})$ is a smooth (but not necessarily convex) function. The most common specification is that Ξ is an index set of all training samples $\Xi = \{1, 2, \ldots, N\}$ and $F(\boldsymbol{\theta}; \xi)$ is the cost function with respect to the training sample indexed by ξ.

Gradient-descent (GD) is a standard algorithm, employed by many classical machine learning models to minimize the above-mentioned optimization problem. GD minimizes the cost function $F(\boldsymbol{\theta})$ by executing the following two steps *iteratively*.

- *Gradient step.* This step is responsible for computing the gradient (Equation 2.7) corresponding to the cost function $F(\boldsymbol{\theta}; \xi_i)$, based on i^{th} sampled example from the training database, with respect to the model parameters ($\boldsymbol{\theta}$).

$$\boldsymbol{G}(\boldsymbol{\theta}; \xi_i) = \nabla F(\boldsymbol{\theta}; \xi_i) \tag{2.7}$$

- *Descent step.* This step then updates the current model parameters ($\boldsymbol{\theta}$) in a direction opposite to the compute gradient as shown in Equation 2.8. More precisely, given a training database with N training examples and a learning rate γ_k, the model is updated at any given step k using n examples (such that $1 \leq n \leq N$) as follows.

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \gamma_k \cdot \sum_{i=1}^{n} \boldsymbol{G}(\boldsymbol{\theta}; \xi_i) \tag{2.8}$$

---

[1] This overlap is often derived from profiles maintained by users across various web applications along with interconnection mechanisms for cross-system interoperability [36] and cross-system user identification [35].

We note that the above update rule is known as *stochastic* GD update when $n = 1$, then *mini-batch* GD update when $1 < n < N$, and lastly *batch* GD update when $n = N$. Depending on the size of the mini-batch (n), there exists a trade-off between the robustness of a given update (noise in the computed gradient) and the time required to compute this update. Lastly, the initial model parameters $\boldsymbol{\theta}^{(0)}$ typically have a specified value or follow a predefined probability distribution (e.g., Gaussian).

2.6 Trust-distrust Relationship in Online Social Networks

Online Social Networks (OSNs) are becoming increasingly popular nowadays as online places where users gather and exchange information. However, this information exchange also raises severe trust-distrust issues. Trust-distrust relations between users play a vital role in making decisions in OSNs like voting for administrators. In practice, the available explicit trust relations are often extremely sparse, therefore making the prediction task even more challenging. Weighted nearest neighbor algorithms are widely used for predicting trust relations [197, 126]. Algorithm 3 demonstrates one such algorithm leveraging K-nearest neighbors (KNN) to predict trust relations.

We denote the trust level of user w for a user v as R_{wv}. Given n classes with labels C_0, C_1,..., C_n which reflect the different levels of trust or distrust [67] between two users, we define a mapping function ϕ such that $\phi(R_{wv}) = C_i$ and $0 \le i \le n$. We then define $\textsc{Score}(w, v, C_i)$ as follows.

$$\textsc{Score}(w, v, C_i) = \begin{cases} 1 & \phi(R_{wv}) = C_i \\ 0 & \phi(R_{wv}) \neq C_i \end{cases} \tag{2.9}$$

Since trust relation between users is asymmetric, it is possible to have $\textsc{Score}(w, v, C_i) \neq \textsc{Score}(v, w, C_i)$ when $R_{wv} \neq R_{vw}$.

These three phases resemble the ones in Algorithm 1. The first phase (similarity computation) employs the standard cosine similarity between users. The second phase is similar to the one in Algorithm 1 and derives the KNN set for a given user. Finally, the last phase predicts the trust relation between two users based on the KNN graph constructed in the previous two phases.

2.7 Evaluation Metrics

We recall here some standard metrics used to evaluate the quality of recommender systems. Based on the literature of recommender systems [43, 160], we use *Precision*, *Recall*, and *F_1-score* as our metrics to assess the quality of recommenders. Table 2.1 presents the terms needed for defining these metrics: *true positives* (tp), *true negatives* (tn), *false positives* (fp), *false negatives* (fn).

Algorithm 3 Trust Prediction

```
Require: $\mathcal{U}$: User set; $\mathcal{U}_w$: Set of users who trusted/distrusted another user with user-id $w$.
Ensure: $R_{wv}$: Trust level of user $w$ for a user $v$.

Phase 1 - Similarity computation: GetSimilars($v, \mathcal{U}$)

Ensure: $s_v$: Dictionary for user $v$ with user-ids as keys and similarities as values.
for $w$ in $\mathcal{U}$ do
    $s_v[w] = \frac{\sum_{u \in \mathcal{U}_w \cap \mathcal{U}_v} R_{wu} R_{vu}}{\sqrt{\sum_{u \in \mathcal{U}_w} R_{wu}^2} \sqrt{\sum_{u \in \mathcal{U}_v} R_{vu}^2}}$    (2.10)
end for
return: $s_v$

Phase 2 - Neighborhood computation: kNN ($v, \mathcal{U}$)

Ensure: $N_v$: $K$ most similar users to user $v$.
$N_v$ = nLargest($K$, GetSimilars($v, \mathcal{U}$))
return: $N_v$

Phase 3 - Prediction: PredictTrust($w$, $v$)

Ensure: Trust prediction of user $w$ for a user $v$.
return: $\underset{C \in \{C_0, \ldots, C_n\}}{\operatorname{argmax}} \sum_{l \in \mathrm{kNN}(w, \mathcal{U})} \mathrm{Score}(l, v, C)$
```

	Relevant	Irrelevant	Total
Recommended	tp	fp	$tp + fp$
Not Recommended	fn	tn	$fn + tn$
Total	$tp + fn$	$fp + tn$	N

Table 2.1 – *Confusion Matrix for true/false positive/negative recommendations.*

Precision or *True Positive Accuracy (TPA)* is the ratio of the number of relevant recommended items to the total number of recommended items.

$$Precision = TPA = \frac{tp}{tp+fp}$$

Recall or *True Positive Rate (TPR)* is the ratio of the number of relevant recommended items to the total number of relevant items.

$$Recall = TPR = \frac{tp}{tp+fn}$$

F_1*-score* is used to evaluate precision and recall simultaneously. Mathematically, it is the harmonic mean of Precision and Recall.

$$F_1 - score = 2.\frac{Precision.Recall}{Precision + Recall}$$

We use these standard evaluation metrics throughout the rest of the thesis. In some sections of this thesis, we introduce some additional evaluation metrics like Mean Absolute Error (MAE) which are more specific to that section only.

Based on these background concepts, we explore and address the technical challenges for designing a personalization system (i.e., *scalability*, *privacy*, and *heterogeneity*) in the next three parts of this thesis. At the beginning of each part, we summarize the major contributions of that specific part of the thesis.

PART II

Scalability

As the amount of web data increases, the need for highly scalable personalization solutions grows proportionally. In this part of the thesis, we focus on two primary directions that improve the scalability of recommender systems.

- The first one is *democratization* where *customized* computation jobs are *iteratively* offloaded to devices of the end-users which could be either high-end devices like laptops or lightweight mobile devices like smartphones or tablets. The computation jobs are typically lightweight and customized to the computational capacity of the device. We provide democratized solutions for classical collaborative filtering (in §3.1) and demonstrate how it could be extended to classical machine learning (in §3.2).
- The second one is *incrementality* where the personalization model is updated in an *incremental* manner to incorporate freshly arriving data without significant computational overhead. Recommenders typically collect user *preferences* using *explicit* feedback such as *numerical* ratings (star ratings in IMDb, Netflix, Amazon), *binary* preferences (likes/dislikes in Youtube), or *unary* preferences (retweets in Twitter). We provide incremental solutions for recommenders employing the above-mentioned explicit feedback (in §4.1) as well as recommenders using implicit feedback such as sequences of consumed items (in §4.2).

3 Democratization

3.1 HyRec: Towards a hybrid architecture

3.1.1 Overview

The motivation of this work is to explore solutions that can *democratize* personalization by making it accessible to any service provider, without requiring huge investments. We introduce HyRec, a *hybrid* architecture capable of providing a cost-effective[1] scalable personalization platform to any service provider. Instead of scaling through either larger and larger recommendation back-end servers, or through fully decentralizing the recommendation process by relying solely on the front-end clients, HyRec delegates expensive computation tasks to clients while, at the same time, retaining on the server side the system's coordination tasks and the maintenance of the nearest-neighbor graph (for users) which reflects the relationship between different users. In a later section of this chapter, we also demonstrate how to extend this democratization idea to enable service providers to offload various machine learning tasks (e.g., classification, ranking) on mobile devices like smartphones, tablets.

HyRec employs *user-based collaborative filtering* (§2.1.1), namely predicting the interests of a user by collecting preferences or taste information from many other users [60]. CF is content agnostic and represents a natural opportunity for decentralizing recommendation tasks on user devices. More specifically, HyRec adopts a k nearest neighbor (kNN) approach (Algorithm 1), which consists of computing the k nearest neighbors according to a given similarity metric, and identifying the items to recommend from the preferences of these neighboring users [175]. The challenge here is to cope with a large number of users and items. Traditional centralized recommendation architectures achieve this by computing neighborhood information offline and exploiting elastic cloud platforms to massively parallelize the recommendation jobs on a large number of nodes [46, 49]. Yet, offline computation is less effective when new content is being added continuously as well as the dynamic change in user preferences. Forcing periodic re-computations, induces significant costs [46, 121, 133].

---

[1] Cost implies Total Cost of Ownership (TCO).

HYREC's architecture avoids the need to process the entire sets of users and items by means of an *iterative* sampling-based approach inspired by epidemic (gossip-based) computing [183, 22], and successfully used in state-of-the-art k-nearest-neighbor graph construction [54] as well as query processing [3].

The computation for the personalization operations of a user are performed at the browser of that user's machine (which we sometimes simply call *the user* or *the client*). The HYREC server provides each user with a sample set of profiles of other users (*candidate set*). Every user then computes her k nearest neighbors followed by the most popular items preferred by her nearest neighbors. The server uses, in turn, the user's new neighbors to compute the next sample. This iterative process implements a feedback mechanism that improves the quality of the selected neighbors and leads them to converge very quickly to those that could have been computed using global knowledge in an offline manner.

We evaluate HYREC in the context of two use cases. The first is Digg, a personalized feed, whereas the second is MovieLens, a movie recommender. We use real traces in both cases. Our results show that the quality of the KNN approximation provided by HYREC is close to the optimal one. As the convergence of the KNN graph is driven by user activity, users who are frequently online benefit from a better neighborhood than users who are rarely online. We show that the reactiveness of HYREC to compute and refine the KNN during the activity of online users drastically improves the recommendation quality, compared to solutions using offline clustering (which can update this graph after the activity of users) and where personalization is sometimes useless. We also note that user's behavior keeps on changing with time, commonly known as *temporal dynamics*, and hence HYREC, in practice, could lead to better recommendation quality due to incorporation of the recent behavior of the user during the recommendation generation.

3.1.2 HYREC

HYREC lies between fully decentralized, cheap but complex to implement/maintain, and centralized, efficient but potentially costly, recommender frameworks. It leverages the *locality* of the computation tasks involved in user-based CF schemes. In HYREC, (Figure 3.1), when a user accesses a webpage from her browser, the server *(i)* updates the user profile in its global data structure, and then *(ii)* selects a set of candidate users to send to the user (i.e., HYREC client) along with the associated profiles. The client in turn performs the similarity computations between the local profile and the ones of the candidate set followed by the item recommendation. In the following, we briefly describe how the client and the server operate in HYREC.

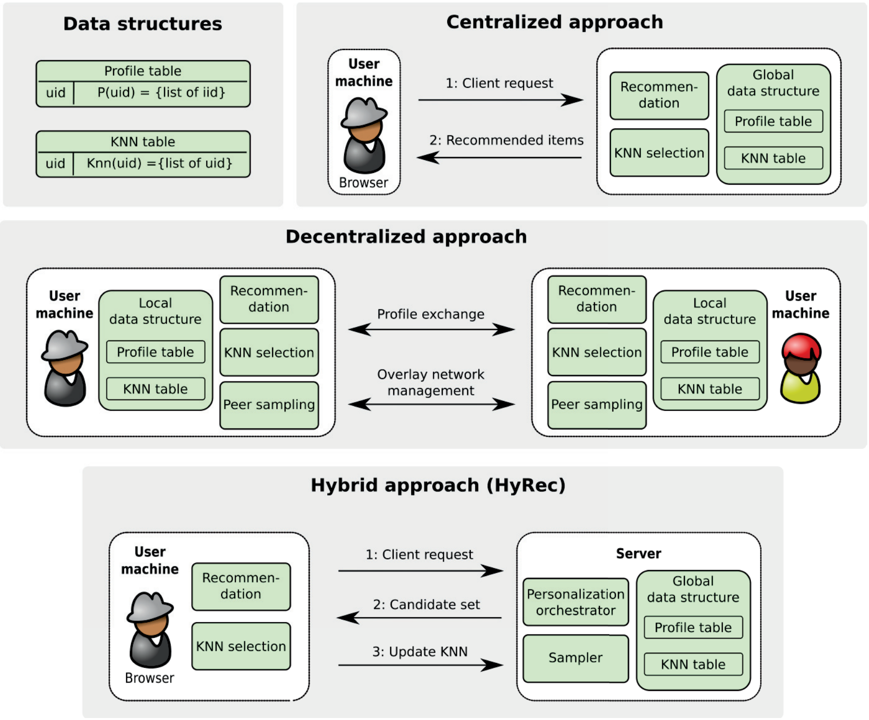


Figure 3.1 – *Centralized, decentralized and hybrid (*HyRec*) architecture of a recommender.*

A. HyRec server

The server is in charge of *(i)* orchestrating the decentralized computations carried out by clients, and *(ii)* maintaining the global data structures, a *Profile table* and a kNN *table*. Each entry in the *Profile* and the kNN tables, indexed by the user-id, contains the user profile and those of its k nearest neighbors respectively.

The server decomposes the recommendation process into *personalization jobs* that run on client-side widgets in the browsers of (connected) users. The kNN selection runs online (as it is achieved by users), and not periodically as usually in a classical centralized architecture, increasing the reactivity of the system. A *personalization job* consists of two tasks: *(i)* a kNN selection task, and *(ii)* an item recommendation task. The HyRec server has two components depicted in Figure 3.1: the *Sampler* and the *Personalization orchestrator*.

Sampler. HyRec relies on a *local* and *iterative* algorithm to associate each user with her k nearest neighbors. We use a sampling-based approach inspired from epidemic clustering protocols [183, 22].

The *sampler* is involved at each iteration of the kNN selection process and provides each client with a small (with respect to the total number of users) set of candidate users, from which the client selects its next k nearest neighbors. Let k be a system parameter determining the size of a user's neighborhood, N_u and containing the k nearest neighbors of u (computed so far). The sampler builds a sample $S_u(t)$ for a user u at time t by aggregating three sets: *(i)* the k current nearest (one-hop) neighbors N_u of u, *(ii)* their k nearest neighbors (two-hop),

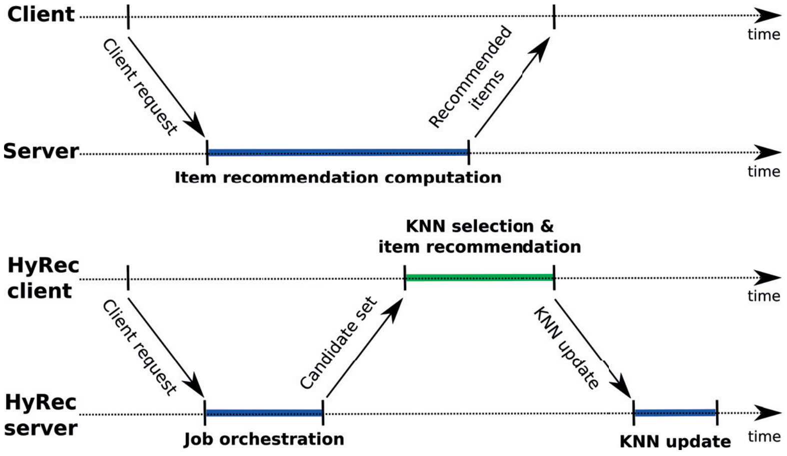


Figure 3.2 – *Timeline: a centralized approach vs.* HYREC

and *(iii)* k random users. Because these sets may contain duplicate entries (more and more as the KNN tables converge), the size of the sample is $\leq 2k + k^2$. However, for a user u, as the neighborhood of u, N_u converges towards the ideal one, N_u^*, the candidate set tends to get smaller as some of u's neighbor share similar neighbors.

By constraining the size of the candidate set, HYREC's sampling-based approach not only limits computational cost, but also network traffic (in terms of bandwidth), while preserving recommendation quality as we show in our experiments. Research on epidemic [183] and k-nearest-neighbor graph construction [54] protocols show that the process converges very rapidly even in very large networks. Using u's neighbors and their neighbors provides the client with a set of candidates that are likely to have a high similarity with u. Adding random users to the sample prevents this search from getting stuck into a local optimum. More precisely, this guarantees that the process will eventually converge in the absence of profile changes by recording the user's k-nearest neighbors in the set N_u, so that $\lim_{t \to \infty}(N_u - N_u^*) = 0$, where N_u^* is the optimal set (*i.e.*, containing the k most similar users). When profiles do change, which happens frequently in the targeted applications (e.g., news feed), the process provides each user with a close approximation of her current optimal neighbors.

Personalization orchestrator. Once a user u accesses the server, (Arrow 1 in Figure 3.1), the orchestrator retrieves a candidate set, parameterized by k from the sampler and builds a personalization job. The personalization job for u consists in building a message that includes u's profile and the profiles of all the candidates returned by the sampler (Arrow 2 in Figure 3.1). Finally, the orchestrator manages the interaction with the HYREC client: sends the personalization jobs, and collects the results of the KNN selection to update the global data structures. Figure 3.2 illustrates the interactions between the clients and the server in HYREC as well as in a centralized approach.

B. HYREC client

In HYREC, users interact with the recommender system through a web interface. The client side of HYREC consists of a Javascript widget, running in the web browser. This widget serves as a web container that interacts with the server's web API. The HYREC client sends requests to the server whenever u requires some recommendations. The server replies by providing a personalization job containing a candidate set along with the associated profiles. Upon receiving the job, the client *(i)* computes locally the recommendation, and *(ii)* runs locally the KNN selection algorithm. Note that the client does not need to maintain any local data structure: the information is provided by the server and garbage collected once the client has computed the new KNN and sent an update to the server.

Recommendation. The client computes u's personalized recommendations as $R_u = \alpha(S_u, P_u)$, where $\alpha(S_u, P_u)$ returns the identifiers of the top-N most popular items among those that appear in the profiles in S_u, but not in P_u. These consist of the most popular items in the S_u to which u has not yet been exposed.

S_u is composed of the profiles of clients in the candidate set: u's neighbors, u's two-hop neighbors, and k random users. By taking into account the items liked by the (one and two hop) neighbors, the item recommendation exploits the opinions of similar users. By also taking into account items from the profile of random users, it also includes some popular items that may improve the serendipity of its recommendations.

In a real application, once the item to be recommended have been identified, they might need to be retrieved from a web server to be displayed in a web page. We omit the process of retrieving the actual content of these items since that is application-dependent.

KNN selection. The client also updates the user's k-nearest neighbors. To achieve this, the KNN algorithm (Algorithm 1) computes the similarity between u's profile and each of the profiles of the users in the candidate set (S_u). It then retains the users that exhibit the highest similarity values as u's new neighbors, $N_u = \text{KNN}(P_u, S_u)$, where $\text{KNN}(P_u, S_u)$ denotes the k users from S_u whose profiles are most similar to P_u according to a given similarity metric (here the cosine similarity). This data is sent back to the server to update the KNN table on the server (Arrow 3 in Figure 3.1).

3.1.3 Evaluation

In this section, we show that HYREC provides good-quality recommendations and reduces cost. We start with a description of the experimental setup. We then study KNN selection, recommendation quality, and the impact on cost.

A. Experimental setup

Platform. We consider a single server hosting all components (front and back-end) and assume an in-memory database. In practice, several machines can be used to implement each component separately to sustain the load at the network level. Yet, this does not affect the outcome of our experiments. We use a PowerEdges 2950 III, Bi Quad Core 2.5GHz, with 32 GB of memory and Gigabit Ethernet, to evaluate the server. To evaluate the client, we use a Dell laptop latitude E4310, Bi Quad Core 2.67GHz with 4 GB of memory and Gigabit Ethernet under Linux Ubuntu.

Datasets. We use real traces from a movie recommender based on the MovieLens (ML) workload [138] and from Digg [52], a social news web site. The ML dataset consists of movie-rating data collected through the ML recommender website during a 7-month period and is often used to evaluate recommenders [46]. For the sake of simplicity, we project ML ratings into binary ratings as follows: for each item (movie) in a user profile, we set the rating to 1 if the initial rating of the user for that item is above the average rating of the user across all her items, and to 0 otherwise. We use the three available versions of this dataset, varying in their number of users, to evaluate the quality of recommendation in HYREC.

The Digg dataset instead allows us to consider a dynamic setting. Digg is a social news website to discover and share content where the value of a piece of news is collectively determined. We collected traces from Digg for almost $60,000$ users and more than $7,500$ items over 2 weeks in 2010. This dataset contains all observed users in the specified period. Table 3.1 summarizes the workload.

Dataset	Users	Items	Ratings
ML1	943	1,700	100,000
ML2	6,040	4,000	1,000,000
ML3	69,878	10,000	10,000,000
Digg	59,167	7,724	782,807

Table 3.1 – *Datasets statistics*

Competitors. We compare the performance of HYREC with that of several alternatives to highlight the benefits and limitations of our approach. For the alternatives, we distinguish two major categories. *Offline solutions* perform KNN selection periodically on a back-end server (Phase 2 in Algorithm 1), while they compute recommendations on demand on a front-end (Phase 3 in Algorithm 1). *Online solutions* perform both KNN selection and item recommendation on demand on the front-end.

Evaluation scheme. To measure recommendation quality, we split each dataset into a training and a test set sorted according to time. The training set contains the first 80% of the ratings while the test set contains the remaining 20%. The goal of the recommender is to recommend to a user as many positively-rated items from the test set as possible.

Evaluation metrics. To measure the effectiveness of HyRec in finding the nearest neighbors in term of interest, we compare the average profile similarity between users and their neighbors, referred to as *view similarity* in the following. We obtain an upper bound on this view similarity by considering neighbors computed with global knowledge. We refer to this upper bound as the ideal or exhaustive kNN in the rest of the evaluation.

For each rating r in the test set, the associated user requests a set of n recommendations ($\Re$). The recommendation quality metric counts the number of positive ratings for which the $\Re$ set contains the corresponding item from the testing set: the higher the better. If a positive rating represents a movie the user liked, this metric counts the number of recommendations that contain movies that the user is known to like.

B. kNN selection quality

To evaluate the quality of the kNN selection provided by HyRec, we replay the activity and ratings of each user over time. When a user rates an item in the workload, the client sends a request to the server, triggering the computation of recommendations. We compare HyRec with the upper bound provided by the ideal/exhaustive kNN.

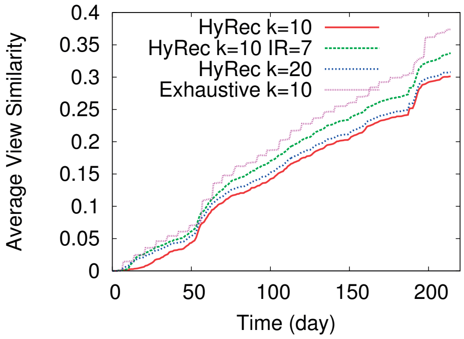


Figure 3.3 – *Average view similarity on ML1 dataset for* HyRec *and ideal* kNN.

Figure 3.3 displays the average view similarity over all the users in the ML1 dataset as a function of time. The plot compares the results obtained by HyRec with those obtained by an offline protocol that computes the ideal kNN once a week. The period of one week allows us to identify a step-like behavior in the offline approach. We observe this behavior in the offline protocol because the neighbors remain fixed between two periodic computations and thus cannot follow the dynamics of user interests. A typical period in existing recommenders is on the order of 24 hours. Such a shorter period would make the steps thinner but it would not lead to faster convergence. Indeed, the upper bound on view similarity can be obtained by connecting the top-left corners of the steps in the offline-ideal (i.e., exhaustive) curve. This corresponds to online protocol that computes the ideal kNN for each recommendation.

Overall, Figure 3.3 shows that HyRec effectively approximates this upper bound. For a neighborhood size of $k = 10$, HyRec's average view similarity remains within 20% of that of the ideal

KNN at the end of the experiment. The curve for $k = 20$ shows the impact of the neighborhood size: larger values of k result in larger candidate sets that converge faster to the nearest neighbors.

HyRec is an online protocol in the sense that it runs KNN selection as a reaction to user requests. The timing of such requests follows the information available in the data trace. As a term of comparison, we also consider a variant (IR=7) that bounds the inter-request time (*i.e.*, the interval between two requests of the same client) to one week. Results show that the quality of KNN selection drastically improves according to the activity of users: more frequent user activity results in better view quality. An inter-request period of one week for $k = 10$ is enough to bring HyRec's approximation within 10% of the upper bound at the end of the experiment.

The iterative approach of HyRec refines its KNN selection over time. As the KNNs of each user converge, the average size of the candidate set tends to decrease as each candidate is more likely to be an actual neighbor. Figure 3.4 depicts the average candidate-set size on the entire ML1 workload as a function of time for different values of k. We observe that the candidate-set size quickly converges to a stable value. For instance, for $k = 10$, its value quickly converges to around 55 instead of the upper bound of 120 (due to $k^2 + 2k$). The small fluctuations in the curve result from the continuous arrival of new users, who start with large candidate sets.

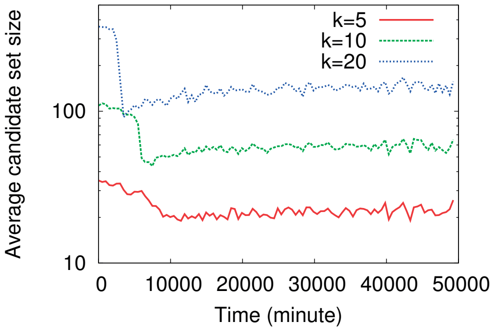


Figure 3.4 – *Convergence of the candidate set size (ML1 dataset).*

C. Recommendation quality

The recommendation process leverages the KNN selection to identify the items to recommend. Figure 3.5 displays the recommendation quality provided by HyRec and by systems based on ideal KNN (both offline and online variants). Results show that the recommendation quality of offline approaches drastically changes according to the period of offline KNN selection (parameter p on Figure 3.5). The online ideal solution, which computes the ideal KNNs before providing each recommendation, provides an upper bound on recommendation performance.

HyRec improves the recommendation quality by 12% with respect the offline ideal approach even when this one runs with a period of 24 hours, which is already more costly than HyRec.

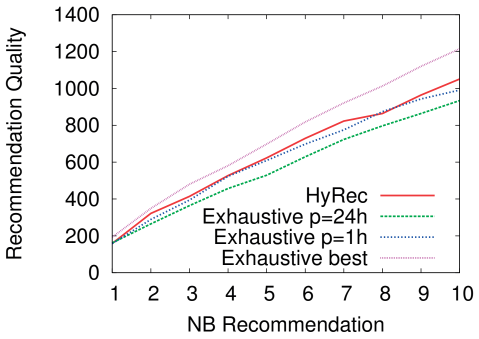


Figure 3.5 – *Recommendation quality on the ML1 dataset for* HyRec *as well as offline and online ideal* KNN *($k = 10$).*

It also provides better performance than offline ideal with a period of 1 hour and scores only 13% below the upper bound provided by online ideal.

To understand HyRec's improvement on offline approaches, consider a user whose rating activity fits inside two updates of offline KNN selection. This user will not benefit from any personalization with an offline approach. This is especially the case for new users which start with random KNNs. In HyRec, on the other hand, users start to form their KNN selection at their first rating and refine it during all their activity. This allows HyRec to achieve personalization quickly, efficiently, and dynamically.

D. Impact on cost

We now compare the cost of running the HyRec front-end with that of running several offline solutions based on the centralized recommender architecture as depicted in Figure 3.1. In such solutions, a front-end server computes the item recommendation in real time upon a client request, while a back-end server periodically runs the KNN selection. Since HyRec leverages user machines to run the KNN selection task, it significantly reduces the cost of running a recommender system.

To ensure a fair comparison, we first identify a baseline by selecting the least expensive offline solution among several alternatives running on Grid5000 [24]. *Exhaustive* is the offline approach we considered earlier. It computes similarities between all pairs of users thereby yielding the ideal KNNs at each iteration. CRec is an offline solution that uses the same algorithm as HyRec (i.e., a sampling approach for KNN) but with a map-reduce-based architecture. Both exploit an implementation of the MapReduce paradigm on a single 4-core node [151]. Finally, *Mahout* and *ClusMahout* are variants based on the user-based CF implementation in Mahout, an open-source machine-learning Apache library [129]. Both exploit the Apache Hadoop platform [80] to parallelize the KNN selection on multiple cores. *Mahout* runs on a single 4-core node, while *ClusMahout* runs on a cluster with two 4-core nodes. Because all four solutions share the same front-end, we only compare the running time of their KNN selection tasks on the back-end. In all cases, we consider two periods for offline KNN selection:

48 hours on MovieLens and 12 hours on Digg.

Figure 3.6 depicts the results. Not surprisingly, we observe a strong correlation between the size of the dataset (in terms of number of users and size of the profile) and the time required to achieve KNN selection. We observe that *CRec* consistently outperforms other approaches on all datasets with the exception of ClusMahout using two nodes on the ML1 dataset. On average, *CRec* reduces the KNN-selection time by 95.5% and 66% with respect to Exhaustive and ClusMahout, respectively. Moreover, the gap between the wall time required by *CRec* and by the other alternatives increases with the size of the dataset. We therefore select *CRec* as a baseline to evaluate the gains provided by HYREC in terms of cost.

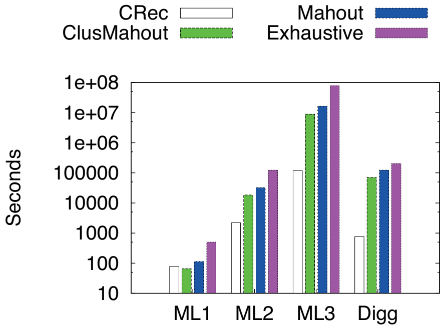


Figure 3.6 – *Time to compute the k nearest neighbors on ML and Digg workloads.*

Specifically, we gauge the cost associated with running *CRec* and the HYREC front-end on a cloud infrastructure using Amazon EC2 services [9]. For the front-end server of both solutions, we consider the cheapest medium-utilization reserved instances which cost around $681 per year (the *Profile table* as well as the KNN *table* need to be stored in memory in order to answer the client requests as fast as possible). For the back-end server of *CRec*, we consider one of the midrange compute-optimized on-demand instances with a price of $0.6 per hour (on-demand instances allow the content provider to be flexible in operating the offline KNN selection task). The efficiency of *CRec*'s KNN selection depends on the frequency at which it is triggered: a higher clustering frequency improves recommendation but it makes more frequent use of the on-demand instances, thereby increasing cost.

Based on these estimates, Table 3.2 summarizes the cost reduction achieved by HYREC as the percentage of the total cost saved by the content provider. We do not consider extra costs for data transfer as the bandwidth overhead generated by HYREC is small and does not exceed the free quota even with the ML3 dataset. Results show that the cost reduction ranges from 9.4% for ML1 with a KNN selection period of 48 hours to 97% for ML3. To compute this last value of 97%, we considered a compute-optimized reserved instance over one year, which is cheaper than the number of required on-demand instances.

Dataset	48h	24h	12h
ML1	9.4%	18.8%	37.7%
ML2	45%	91%	97%
ML3	97%	97%	97%

	12h	6h	2h
Digg	2.6%	5.3%	10.5%

Table 3.2 – *Impact on the cost of a centralized back-end server according to the* KNN *selection period.*

3.1.4 Related Work

A radical way to address scalability is through a significant departure from centralized (cloud-based) architectures, namely through fully distributed CF solutions [190, 155, 11, 136, 195]. While appealing, these solutions face important deployment challenges. They require users to install specific software that must manage their online/offline patterns, while taking care of synchronization between multiple devices that may not be online at the same time. These distributed solutions are also significantly limited in their scalability due to communication overheads across the nodes in the distributed setup. This limitation, combined with the inherent scalability of decentralized solutions, provide a strong motivation for a hybrid approach like ours: namely, combining a centralized entity that coordinates tasks and handles the connections and disconnections of users with processes performing the actual tasks on the clients.

Hybrid approaches have already proved successful in various contexts. SETI@home [162] leverages machine of volunteers for analyzing radio telescope data whereas Weka [104] does something similar for data mining. A distributed Weka requires either a grid hosted by the service provider, or an application server on the clients. In addition, Weka is oriented towards data analysis and does not provide a real-time personalization system. TiVo [5] proposes a hybrid recommendation architecture similar to ours in the context of item-based CF (Algorithm 2). Yet, TiVo does not completely decentralize the personalization process. It only offloads the computation of item recommendation scores to clients (Phase 3 in Algorithm 2). The computation of the correlations between items is achieved on the server side (Phase 1 and Phase 2 in Algorithm 2). Since the similarity computation operation is extremely expensive, TiVo's server only computes new correlations every two weeks, while its clients identify new recommendations once a day. This makes TiVo unsuitable for dynamic websites dealing in real time with continuous streams of items. HyRec addresses this limitation by delegating the entire filtering process to clients: it is to our knowledge the first system capable of doing so on any user-based CF platform.

3.1.5 Conclusion

We report in this work on the design and evaluation of HyRec, a user-based collaborative filtering system. The architecture of HyRec is hybrid in the sense that it lies between traditional centralized systems on the one hand, and fully decentralized P2P solutions on the other.

HyRec seeks to provide the scalability of P2P approaches while retaining a centralized orchestration. We show that HyRec is cost-effective as it significantly reduces the recommendation cost and improves scalability with respect to centralized (possibly cloud-based) solution.

The motivation underlying HyRec is to explore solutions that could in some sense democratize personalization by making it accessible to any service provider company without requiring huge investments. HyRec is generic and can operate in many contexts. In its current version, it performs a user-based CF scheme. However, any data filtering algorithm which can be split through the browsers of users can be used. We also experimented for instance with an item-based CF recommendation protocol (Algorithm 2). In this implementation, the server provides the client browsers with the current item (i.e., the item currently viewed by the user) and a candidate set containing the neighborhood of the current item and their associated profiles. Here, the profile of an item is the set of users exposed to the items and their associated ratings. The item recommendation process, executed at the user's machine, computes the nearest items (i.e., in term of user interest) to the current item. Whereas the recommendation quality is smaller than the user-based CF variant described in this work, the same behavior is observed regarding the kNN selection: the neighborhood of popular items is refined better than unpopular items which is attributed due to the activity difference in the popular and unpopular items.

Lastly, we note an important aspect in HyRec which is the Quality-of-Service as ultimately perceived by the end user. With a good Internet connection and a powerful device, a user will get its recommendations much faster than a user with a poor connection and an old device. However, as the Javascript widget of HyRec is totally asynchronous, the delay to display the recommendations does not block the display of the rest of the web page. With the advent of Web 2.0 applications, end users' resources become exploitable transparently by the service provider even through multi-threading Javascript tasks attached to web pages [98]. This new feature increases the high potential of hybrid approaches as HyRec.

The possibility of attacks and their potential impact can also be a determining factor to decide whether to deploy or not a hybrid architecture in practice. Indeed, HyRec limits the impact of untrusted and malicious nodes: each user computes only its own recommendations. However, it is also possible to use privacy-aware mechanisms such as homomorphic encryption [88] or differential privacy [55] to generate encrypted or differentially-private profiles of the users. Then, these private profiles are offloaded and used for the recommendation computations.

3.2 Extension to machine learning on mobile devices

We now provide a brief overview regarding how the underlying idea of HyRec can be easily extended to other context like *machine learning* on clients' devices (typically mobile devices). In this extension, we propose a framework named HyML, similar to HyRec, which offloads machine learning tasks to mobile devices. HyML offloads the training phase (Equation 2.7 in §2.5) of any GD-based machine learning model to the mobile devices.

Similar to HYREC, our HYML framework also enables any service provider to deploy large-scale ML applications without requiring huge investments. We focus on ML applications used by clients through mobile devices (e.g., smartphones, tablets) of which the number is increasing rapidly. Mobile devices provide a perfect opportunity since the number of smartphones users is rising significantly (currently 5 billion). Furthermore, several big industrial players such as Huawei are focused on increasing the computational capacity of mobile devices by introducing chips with performance in the order of tens of teraflops (trillion floating point operations per second).

Following HYREC's approach, the service provider offloads the most computational task (i.e., model training via gradient computation) to the client's mobile device which is considered as a computation unit in this work. However, the service provider must ensure that the effect on the client's device in terms of latency or energy consumption (i.e., less workload) is negligible while also accelerating the learning process with the huge amount of incoming data. Hence, there is an underlying trade-off between these two objectives, from the service provider's and the client's perspectives, depending on the size of workload to be offloaded to the devices.

Due to the asynchronous nature [120] of the training procedure (i.e., model updates via gradients) combined with the heterogeneity of the mobile devices, there exists significant performance difference between the slow and fast mobile devices. HYML currently employs classical heterogeneity-aware model update algorithms [97, 196] in such a heterogeneous environment of mobile computing devices.

A. HYML Overview

HYML is a distributed framework that enables the service provider to employ mobile devices as workers. The design of HYML is suitable for the deployment of any ML algorithm in which the workers compute updates based on a current model version and a centralized server generates a new model version by using these updates e.g., gradient-descent (Equations 2.7 and 2.8 in §2.5). A key component of HYML is a *smart sampler*, that employs an ML algorithm (e.g., regression) to ensure that the workload for each device is proportional to the device capabilities. This smart sampler handles the aforementioned trade-off based on the size of the workload to be offloaded to the mobile devices.

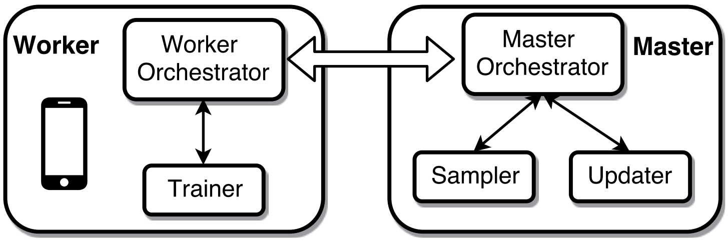


Figure 3.7 – *The architecture overview of* HYML.

HYML, as shown in Figure 3.7, has a classical master-worker architecture where the service provider hosts the master module and each mobile device hosts the worker module. Below, we briefly describe the functionality of each module of our framework.

Master. HYML's master component is similar to the Server component in HYREC (Figure 3.1). More specifically, the master component in HYML consists of three subcomponents which we describe briefly in the following.

1. *Master-orchestrator (*MORC*).* This subcomponent is similar to the *Personalization orchestrator* component in HYREC and is responsible for the complete orchestration of the model update process in HYML. The MORC enables the communication between the master and the workers (i.e., the mobile devices). Whenever a worker makes a query to the master, the MORC responds back with the customized workload, i.e., the current model (cached in the updater subcomponent) along with a mini-batch which is provided by the sampler (depending on the predicted mini-batch size). The MORC also forwards the computed gradients, received from the workers, to the updater subcomponent.
2. *Sampler.* This subcomponent is similar to the *sampler* component in HYREC and is responsible for generating the workload to be sent to the worker. For each user query, the sampler first predicts the appropriate mini-batch size depending on the device features by employing any classical regression technique and hence handles the computation workload to be offloaded to the client device. We highlight that HYML employs a dynamic workload whereas HYREC employs a constant workload, dependent on the number of neighbors (K) in the nearest-neighbor graph, for any device. In this regard, HYREC was designed to offload computational tasks to browsers of users and hence a constant workload is practical (limited by the browser cache size). Lastly, the sampler also generates a mini-batch sample (from the cached dataset[2]) based on the predicted size. The workload consists of this generated mini-batch along with the model which HYML sends to the worker.
3. *Updater.* The updater component is responsible for caching the model and performing the model update (descent step in Equation 2.8 in §2.5) based on the gradients that the MORC forwards. This update operation is comparable to the nearest-neighbor update performed in the KNN table in HYREC by the Server component.

Worker. The worker performs the computationally demanding part of the model training, namely the gradient computation ($\boldsymbol{G}(\boldsymbol{\theta})$ in Equation 2.7), thus mitigating the requirement for huge investments on cloud resources. This component is similar to the *browser* component in HYREC (Figure 3.1). The worker consists of two main subcomponents described as follows.

1. *Worker-orchestrator (*WORC*).* The WORC subcomponent enables the worker to communicate with the master and also initiates the communication with the master by forwarding any client query like an image classification. It receives the computation workload from the MORC and then invokes the trainer subcomponent to employ this workload for performing local on-device training. The computed gradients are sent back to the MORC along with the performance statistics of the mobile device during the training step for improving the

---

[2] The cached dataset could be appended with new examples either collected from the clients or public sources.

accuracy of the sampler.

2. *Trainer.* The trainer component is responsible for computing the gradients (as explained in §2.5) employing the model and the mini-batch received from the WORC. This component is similar to the KNN selection one in HYREC.

B. Performance impact of mobile devices

As we mentioned before, HYML employs mobile devices as workers and offloads the computations to these devices from the service provider. However, classical distributed ML frameworks like Tensorflow, DL4J or Torch typically employ CPUs or GPUs as their computation units. Hence, we compare the throughput of mobile devices with the throughput achieved by CPUs or GPUs. For the sake of fairness, we deploy DL4J on a mobile device (using HYML), a single node consisting of 2 CPUs (Intel Xeon E5-2620) and 1 GPU (Nvidia Titan Black), as well as a Spark cluster with 8 nodes of similar configuration.

We use a classical Convolutional Neural Network (CNN [112]), using mini-batch of 100 examples, as the classifier on a dataset consisting of hand-written characters and digits EMNIST[3] where each training instance has 784 input features. Figure 3.8 compares the throughput (i.e., number of model updates per minute) among the various setups (i.e., mobile devices, CPU, GPU, Spark cluster) employing the CNN classifier. Interestingly, we observe that a GPU (Nvidia Titan Black) achieves 42 times higher throughput than a mobile worker (Honor 9) in our current setup. This comparison gives us a nice estimation of the number of mobile workers required to replace one node in any classical distributed framework in order to achieve the same throughput. Lastly, we also note that the scalability saturates with an increasing number of nodes (as observed from the Spark cluster) due to the communication overhead [166, 192].

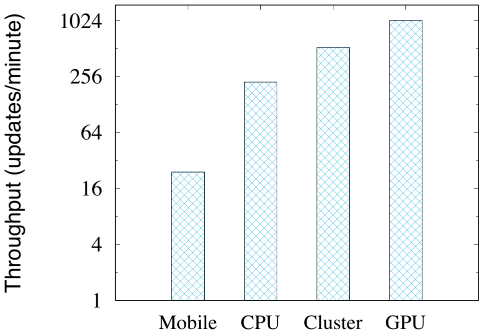


Figure 3.8 – *Throughput comparison among single nodes (mobile, CPU, GPU) and a cluster.*

C. Conclusion

We demonstrate how the approach of HYREC could be extended to machine learning on user devices leading to HYML. HYML democratizes the machine learning task to the mobile devices by offloading the computation-intensive training part to the devices. Moreover, our

---

[3] https://www.nist.gov/itl/iad/image-group/emnist-dataset

GD-based approach could be extended to other machine learning algorithms as well for e.g., expectation-maximization.

4 Incrementality

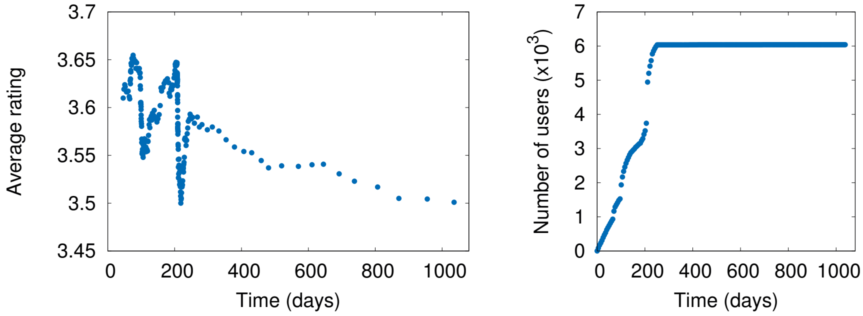
In this chapter of the thesis, we tackle the scalability problem by updating the recommender system in an incremental manner with fresh incoming data. We call this approach towards addressing scalability as *incrementality*. We present I-SIM in §4.1 to incorporate explicit feedback incrementally whereas we introduce CIP in §4.2 to handle implicit feedback in an incremental manner (for online platforms without explicit feedback).

4.1 I-SIM: Incremental Similarity

4.1.1 Overview

The starting point of this work is the observation that existing similarity metrics were not designed to handle a very large number of users with rapidly changing behavior. The number of recommendation requests issued by users today, is in the order of millions per day [92], which poses a major scalability challenge. State-of-the-art scalable recommenders [118, 158, 15] employ batch processing and update their recommenders at intervals of weeks. They indeed achieve low latency recommendations, but ignore the temporal behavior of users (*temporal relevance* [110, 122]), thereby leading to relatively lower recommendation accuracy. For example, the number of views of news articles saturates within a few hours [117]: these articles should be recommended within this time span to be relevant. On the other hand, the very few recommenders that account for temporal relevance [92, 110] do not scale as they require heavyweight computations, inducing high energy consumption which is becoming a key issue in cloud computing [12].

An interesting temporal effect that emerges from the MovieLens (ML) dataset [138] is depicted in Figure 4.1. Users typically provide their preferences for items in terms of *feedback* like *ratings*. Figure 4.1(a) conveys the fact that the moving global average rating fluctuates within the first 200 days. This fluctuation can be attributed to the initial user churn (as shown in Figure 4.1(b)). However, when the number of users is stable, we observe a downward trend in the average rating which saturates at around 3.5. The primary reasons behind this temporal



(a) *Moving global average rating where each point averages the 100,000 previous ratings.*

(b) *Total number of users.*

Figure 4.1 – *Temporal effects in ML-1M dataset.*

behavior can be attributed to the users' *preference* and *behavioral drifts*.

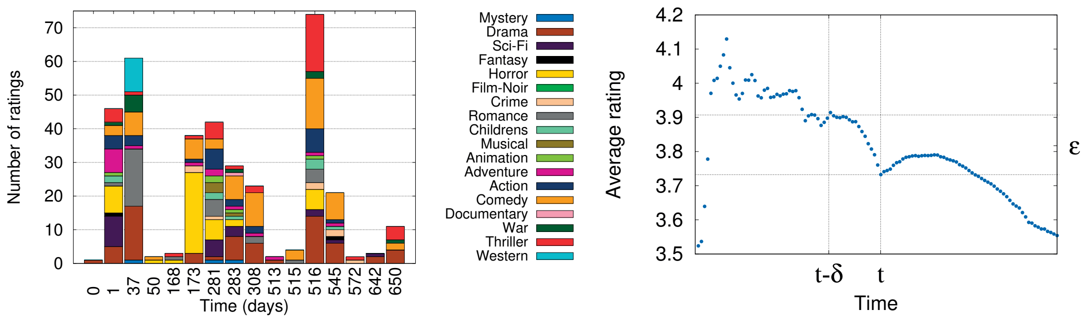
Preference drift. Users' *preferences* typically fluctuate over time. For example, a change in the family structure can drastically change shopping patterns. Figure 4.2(a) depicts the preference distribution of an individual user over time. The top genre preferences for this user on Day-1 were Adventure, Horror and Sci-Fi whereas on Day-37 her preferences were mostly Western, Romance and Drama. We also observe other genre preferences that vary over the following days (e.g. Thriller).

Behavioral drift. At another personalization level, a user's feedback (e.g. scores, ratings, votes) also fluctuates over time possibly due to her varying behavior (e.g. mood). This feedback fluctuation results in a *user bias*. Given that a user u provides a feedback s_{ui} for an item i at a time $t-\delta$ when her average feedback was $\bar{s_u}(t-\delta)$, then the user is biased towards this item by $b_{ui}(t-\delta) = s_{ui} - \bar{s_u}(t-\delta)$. Sarwar et al. empirically showed that including such a user bias in the similarity computations, however in a static (non-temporal) manner, leads to better recommendation quality [157]. The change in this user bias $(b_{ui}(t-\delta) - b_{ui}(t))$ over time is the change in the average feedback $(\bar{s_u}(t) - \bar{s_u}(t-\delta))$.

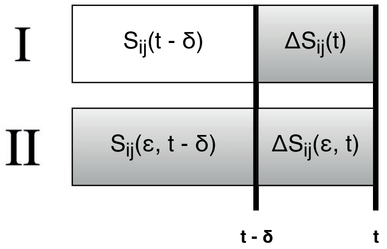
Figure 4.2(b) captures the change in the user bias (behavioral drift) which we quantify using a key user attribute (ϵ) defined as follows: *the average feedback of a user varies over time in steps of a temporal parameter* ϵ*, also denoted by* $\epsilon(t)$*, between a time interval* $[t-\delta, t]$. State-of-the-art incremental similarity metrics [122, 92] do not take into account this attribute (Figure 4.2(c)I). Performing incremental updates based on the temporal parameter ϵ is non-trivial. Similarities until time $t-\delta$ are also a function of ϵ and thus also need to be adjusted at time t (Figure 4.2(c)II).

Based on these observations, one can easily infer that users' temporal behavior can impact the prediction accuracy significantly. However, designing an incremental similarity metric that captures this temporal behavior is non-trivial.

Contributions. The main contribution of this work is a novel similarity metric, we call I-SIM, which enables lightweight similarity computations incorporating the preference and



(a) *Preference drift of a user in ML-1M.* **(b)** *Behavioral drift of a user in ML-1M.*


I
II
$S_{ij}(t - \delta)$
$\Delta S_{ij}(t)$
$S_{ij}(\varepsilon, t - \delta)$
$\Delta S_{ij}(\varepsilon, t)$
t - δ
t

(c) *Dependence on temporal parameter ϵ.*

Figure 4.2 – *Limitations of state-of-the-art similarity metrics with respect to temporal relevance and incremental updates. The gray areas in the right subfigure indicate the similarities (S_{ij}) that need to be updated within a time interval $[t-\delta, t]$.*

behavioral drifts. I-SIM can be considered as a “temporalization” of the adjusted cosine similarity [157] and hence of the cosine similarity. Therefore, I-SIM can be easily integrated with time-aware applications in OSNs. In this work, we primarily focus on collaborative filtering but nonetheless we also explore trust predictions in OSNs.

I-SIM is *lightweight* in the sense that it can be updated incrementally to achieve low latency and limited energy consumption. In particular, I-SIM accounts for temporal relevance through an exponential decrease in the weight of previous feedback over time. We formally prove that the time complexity[1] of I-SIM is $\mathcal{O}(|\Delta U|)$ where ΔU is the set of active users within a given time interval (unlike the time complexity of non-incremental metrics [157] which is $\mathcal{O}(|U|)$ where U is the set of total users in the system).

First, we illustrate the power of I-SIM in personalization applications by implementing a novel recommender leveraging I-SIM, which we call SWIFT (**S**calable **I**ncremental **F**lexible **T**emporal recommender). SWIFT is interesting in its own right, as it enables flexible switching between *stream processing* and *batch processing* [163]. We demonstrate the efficiency of I-SIM through an in-depth experimental evaluation of SWIFT. More precisely, we compare SWIFT with recommenders using incremental similarity computations (TENCENTREC [92]), matrix factorization techniques using temporal relevance (TIMESVD [110]), Alternating Least Squares (ALS [111]) and factored similarity models (FISM [99]), on real-world traces in terms of latency, energy consumption, and accuracy.

---

[1] If not stated otherwise, we refer to the worst-case complexity.

Second, after demonstrating that trust relations in OSNs exhibit temporal behavior, we illustrate the power of I-SIM for trust-distrust predictions in OSNs by implementing I-TRUST. We empirically show that I-TRUST significantly outperforms the non-incremental alternative, both in terms of runtime and accuracy.

4.1.2 I-SIM: A Novel Similarity

In this section, we first pose the similarity computation problem more formally and then present our I-SIM similarity metric before analyzing it. We then show how I-SIM enables incremental updates (for item-item similarities) over time.

A. Problem Definition

Let $\mathcal{U}$ be a set of users, $\mathcal{I}$ be a set of items, and $S_{ij}(t)$ be the similarity between items $i, j \in \mathcal{I}$ till timestep t. We define the similarity function as follows.

$$S_{ij}(t) = \frac{P_{ij}(t)}{\sqrt[n]{Q_i(t)} \cdot \sqrt[n]{Q_j(t)}} \tag{4.1}$$

where n is a positive integer, P is a function of the item vectors i, j, and Q is a function of each individual item vector. For example, if we take the standard cosine similarity (Equation 2.10), then n is 2, P is the dot product of item vectors i and j whereas Q is the squared L^2-norm of each individual item vector. Note that the similarity function definition is formulated for the similarity metrics designed for sparse data (e.g. cosine, jaccard, pearson correlation). For sparse data, which often contains asymmetric data, similarity depends more on attributes that are shared, rather than attributes that are lacking.

For an incremental similarity computation, each of these terms (P, Q) could be incrementally updated as follows.

$$P_{ij}(t) = \Delta P_{ij}(t) + P_{ij}(t-1)$$
$$Q_i(t) = \Delta Q_i(t) + Q_i(t-1)$$

This incremental update seems straightforward when each of the P and Q functions could be expressed as a summation term independent of any time-varying parameter (Figure 4.2(c)I). Nevertheless, for more precise similarity metrics, like adjusted cosine similarity, each timestep depends on some time-varying parameter like the average rating of users. Therefore, the P and Q values, computed in all previous $t-1$ timesteps, need to be updated (Figure 4.2(c)II).

In this work, we solve this non-trivial problem by essentially caching some additional terms. We break the update computation into two components: *standard* (P^s, Q^s) and *adjustment*

(P^a, Q^a) components as follows.

$$P_{ij}(t) = \underbrace{P^s_{ij}(t)}_{\text{standard component}} + \underbrace{P^a_{ij}(t)}_{\text{adjustment component}}$$

$$Q_i(t) = \underbrace{Q^s_i(t)}_{\text{standard component}} + \underbrace{Q^a_i(t)}_{\text{adjustment component}}$$

More precisely, the standard component incorporates the preference drift (Figure 4.2(a)) whereas the adjustment component incorporates the behavioral drift (Figure 4.2(b)).

B. I-SIM

We now describe our I-SIM metric which temporalizes adjusted cosine similarity (Equation 2.3). Given m items and n users, the overall time complexity of the similarity update for standard techniques (Algorithm 2) is $\mathcal{O}(m^2 n)$ per timestep. Naively augmenting the standard adjusted cosine with temporal relevance would require computing item-item similarities at each batch update leveraging all the ratings (Figure 4.2(c)II). The resulting time complexity ($\mathcal{O}(m^2 n)$ per batch update) would be prohibitive for an online recommender.

We first rewrite the adjusted cosine similarity (Equation 2.3), incorporating temporal relevance (Equation 2.5), in terms of *pre-normalized correlation* (P_{ij}) and *normalization factors* (Q_i, Q_j) following the pattern presented in Equation 4.1.

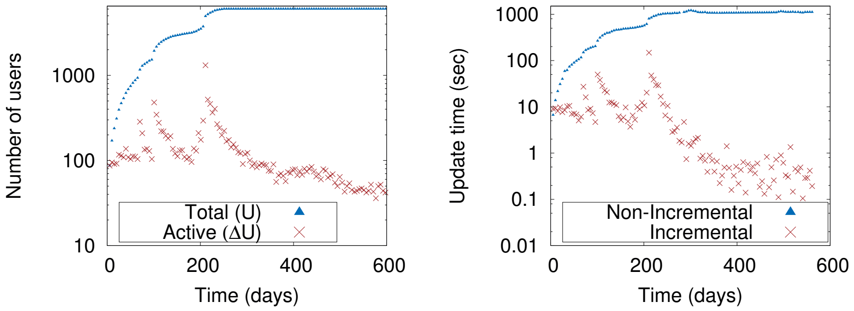
$$S_{ij}(t) = \frac{P_{ij}(t)}{\sqrt{Q_i(t)}\sqrt{Q_j(t)}} \tag{4.2}$$

where

$$P_{ij}(t) = \sum_{u \in \mathcal{U}^t_i \cap \mathcal{U}^t_j} f^{\alpha}_{ui}(t)(r_{ui} - \bar{r_u}(t)) f^{\alpha}_{uj}(t)(r_{uj} - \bar{r_u}(t)) \tag{4.3}$$

$$Q_i(t) = \sum_{u \in \mathcal{U}^t_i} (f^{\alpha}_{ui}(t)(r_{ui} - \bar{r_u}(t)))^2 \tag{4.4}$$

Next, we show that the functions $P_{ij}(t)$ and $Q_i(t)$ can be incrementally updated with a time complexity $\mathcal{O}(|\Delta\mathcal{U}|)$. Thus $S_{ij}(t)$ can also be incrementally computed on-the-fly. Additionally, this incremental feature reduces the time complexity drastically, enabling lightweight model updates with incoming streams of data. The *active users* at any given time interval are the users who provide ratings in that interval. Figure 4.3(a) compares the total number of users ($|U|$) at any given time with the number of active users ($|\Delta U|$) during the last 5 days. Figure 4.3(b) indicates that the computation time required for the similarity update of our incremental approach on a single machine is a few orders of magnitude lower than a non-incremental one. We also observe that the computation time for the incremental approach (Figure 4.3(b)) corresponds to the number of active users (Figure 4.3(a)) at any given time.



(a) *Total users vs active users.* **(b)** *Similarity computation time.*

Figure 4.3 – *Comparison between incremental (*I-SIM*) and non-incremental similarity computations [157, 5] for ML-1M dataset. The time interval for the active users is* 5 *days.*

Before providing the incremental update relations, we introduce two adjustment terms (L, M). These adjustment terms incorporate the behavioral drift captured by $\epsilon(t)$.

$$L_{ij}(t) = \sum_{u \in \mathcal{U}_{ij}^t} \epsilon(t) f_{ui}^{\alpha}(t) f_{uj}^{\alpha}(t)[(r_{ui} - \bar{r_u}(t)) + (r_{uj} - \bar{r_u}(t))],$$

$$L_i(t) = 2 \sum_{u \in \mathcal{U}_i^t} \epsilon(t) f_{ui}^{2\alpha}(t)(r_{ui} - \bar{r_u}(t)) \tag{4.5}$$

$$M_{ij}(t) = \sum_{u \in \mathcal{U}_{ij}^t} \epsilon(t)^2 \cdot f_{ui}^{\alpha}(t) f_{uj}^{\alpha}(t),\ M_i(t) = \sum_{u \in \mathcal{U}_i^t} \epsilon(t)^2 \cdot f_{ui}^{2\alpha}(t) \tag{4.6}$$

where $\epsilon(t) \triangleq \bar{r_u}(t) - \bar{r_u}(t-1)$.

Theorem 1 (P_{ij} INCREMENTAL UPDATE)**.** *Let $\Delta\mathcal{U}_i^t$ denote the set of users who newly rated i at timestep t, i.e. $\Delta\mathcal{U}_i^t = \mathcal{U}_i^t \setminus \mathcal{U}_i^{t-1}$, then the time complexity for updating $P_{ij}(t)$ is $\mathcal{O}(|\Delta\mathcal{U}_i^t| + |\Delta\mathcal{U}_j^t|)$.*

Sketch. The incremental update relation of P_{ij} is:

$$P_{ij}(t) = \Delta P_{ij}(t) + e^{-2\alpha}[P_{ij}(t-1) - L_{ij}(t-1) + M_{ij}(t-1)]$$

where $\Delta P_{ij}(t)$ is defined as follows.

$$\begin{aligned}\Delta P_{ij}(t) = & \sum_{u \in \Delta\mathcal{U}_i^t \cap \mathcal{U}_j^{t-1}} (r_{ui} - \bar{r_u}(t)) f_{uj}^{\alpha}(t)(r_{uj} - \bar{r_u}(t)) \quad + \sum_{u \in \mathcal{U}_i^{t-1} \cap \Delta\mathcal{U}_j^t} f_{ui}^{\alpha}(t)(r_{ui} - \bar{r_u}(t))(r_{uj} - \bar{r_u}(t)) \\ & + \sum_{u \in \Delta\mathcal{U}_i^t \cap \Delta\mathcal{U}_j^t} (r_{ui} - \bar{r_u}(t))(r_{uj} - \bar{r_u}(t))\end{aligned}$$

The summation terms in $\Delta P_{ij}(t)$ have a time complexity of $\mathcal{O}(|\Delta\mathcal{U}_i^t| + |\Delta\mathcal{U}_j^t|)$. The full proof is provided in Appendix §8.1 for interested readers.

□

Note that if $P_{ij}(t)$ was updated non-incrementally then the time complexity would be $\mathcal{O}(|\mathcal{U}_i^t \cap \mathcal{U}_j^t|)$. With each time step, the number of new ratings for i ($|\Delta\mathcal{U}_i^t|$) tends to be significantly smaller than the total number of ratings for i ($|\mathcal{U}_i^t|$). The difference is huge even for the average case as $|\mathcal{U}_i^t|$ can be of the order of all users in the system (Figure 4.3). For example, following the long tail distribution (Figure 4.13(a)) the popular items (20% of all the items) would be rated by nearly 80% of the users in the system.

Theorem 2 (Q_i INCREMENTAL UPDATE)**.** *Given that $\Delta\mathcal{U}_i^t$ denotes the set of users who newly rated i at timestep t, i.e. $\Delta\mathcal{U}_i^t = \mathcal{U}_i^t \setminus \mathcal{U}_i^{t-1}$, then the time complexity for updating $Q_i(t)$ is $\mathcal{O}(|\Delta\mathcal{U}_i^t|)$.*

Sketch. The incremental update relation of Q_i is:

$$Q_i(t) = \Delta Q_i(t) + e^{-2\alpha}[Q_i(t-1) - L_i(t-1) + M_i(t-1)]$$

where $\Delta Q_i(t)$ is defined as follows.

$$\Delta Q_i(t) = \sum_{u \in \Delta\mathcal{U}_i^t} (r_{ui} - \bar{r_u}(t))^2$$

The incremental term ($\Delta Q_i(t)$) has a time complexity of $\mathcal{O}(|\Delta\mathcal{U}_i^t|)$. Note that the complexity for the non-incremental update is again $\mathcal{O}(|\mathcal{U}_i^t|)$. The full proof is provided in Appendix §8.1 for interested readers. □

Hence, the final incremental relations for the adjusted cosine similarity are as follows.

$$P_{ij}(t) = \underbrace{\Delta P_{ij}(t) + e^{-2\alpha} P_{ij}(t-1)}_{\text{standard component}} - \underbrace{e^{-2\alpha}[L_{ij}(t-1) - M_{ij}(t-1)]}_{\text{adjustment component}} \tag{4.7}$$

$$Q_i(t) = \underbrace{\Delta Q_i(t) + e^{-2\alpha} Q_i(t-1)}_{\text{standard component}} - \underbrace{e^{-2\alpha}[L_i(t-1) - M_i(t-1)]}_{\text{adjustment component}} \tag{4.8}$$

$$L_{ij}(t) = \Delta L_{ij}(t) + e^{-2\alpha}[L_{ij}(t-1) - 2M_{ij}(t-1)] \tag{4.9}$$

$$M_{ij}(t) = \Delta M_{ij}(t) + e^{-2\alpha} M_{ij}(t-1) \tag{4.10}$$

The I-SIM values (S_{ij}) can thus be computed on-the-fly, leveraging the incrementally updated $P_{ij}(t)$ and $Q_i(t)$ values. We only need to store the P, L, M and Q values which requires $\mathcal{O}(m^2)$ space. Unlike classical non-incremental algorithms [157], we require extra storage for the adjustment terms (L, M). The non-incremental algorithms [157, 5] also require $\mathcal{O}(m^2)$ space for storing the item-item similarities. Nonetheless, incremental as well as non-incremental algorithms could benefit from sparse data structures as well as *count sketches* [41] for significantly reducing the storage requirements.

We now provide a variant of I-SIM we call I-SIM$_{\epsilon=0}$ which temporalizes pure cosine similarity. Adjusted cosine similarity leads to a pure cosine one if the average rating ($\bar{r_u}$) is set to 0 in Equation 2.3. More precisely, a lack of behavioral drift leads to L_{ij} and M_{ij} being 0 in Equations 4.7 and 4.8 due to $\epsilon(t)$ being 0. The final incremental relations for pure cosine similarity are as follows and do not require any additional storage due to the absence of adjustment terms.

$$P_{ij}(t) = \Delta P_{ij}(t) + e^{-2\alpha} P_{ij}(t-1) \tag{4.11}$$

$$Q_i(t) = \Delta Q_i(t) + e^{-2\alpha} Q_i(t-1) \tag{4.12}$$

I-SIM also applies to the case of static neighborhood based algorithms (i.e. without using temporal relevance by setting α to 0 in the update equations). Such algorithms are often utilized during the cold-start phase of a system.

4.1.3 I-SIM Applications

A. SWIFT: A Novel Recommender

To illustrate the efficiency of I-SIM, we plug it in a novel recommender we design and implement, called SWIFT (**S**calable **I**ncremental **F**lexible **T**emporal recommender). In the following, we present SWIFT and highlight some optimization techniques that speed up its computations, as we later demonstrate through our evaluations.

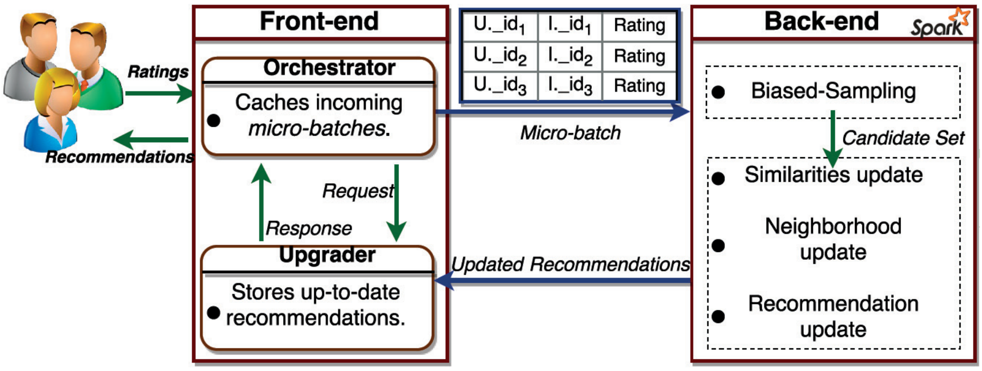


Figure 4.4 – *The architecture overview of* SWIFT.

Framework. As we pointed out, practical recommenders today need to deal with millions of recommendation requests per day, leading to billions of computations. This scale of recommendations calls for a framework which supports the incremental similarity metric that we present in this work. We implement our framework on top of Apache Spark[2] and also choose Apache Cassandra[3] as our storage management system to handle large amount of data.

---

[2] http://spark.apache.org/

[3] http://cassandra.apache.org/

The architecture of SWIFT consists of a front-end and back-end as illustrated in Figure 4.4.

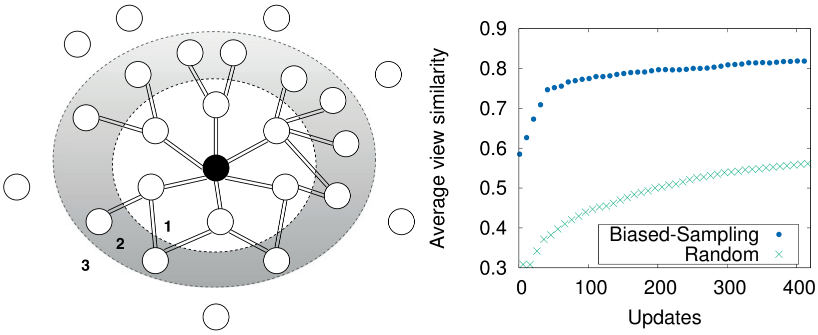
Front-end. The front-end of SWIFT aggregates the new ratings from users in micro-batches. These aggregated micro-batches form the incremental input employed by I-SIM to update the recommender system. The front-end consists of two subcomponents to facilitate the recommendation process.

- *Orchestrator.* This subcomponent is responsible for receiving the recommendation requests from the clients as well as aggregating the incoming rating events into new micro-batches (with pre-defined size) which are temporarily cached on the front-end. The orchestrator also responds to each client's recommendation request by sending the recommendations (received from the back-end and cached in the upgrader). Lastly, the orchestrator periodically transmits the cached micro-batches to the back-end server where the recommender model is updated using I-SIM with these recent micro-batches.
- *Upgrader.* This subcomponent caches locally the most up-to-date recommendations for the clients (received from the back-end server periodically) and later forwards the recommendations to the orchestrator corresponding to the incoming client requests.

Back-end. SWIFT's back-end is responsible for computing the similarity updates for the incoming micro-batches. The back-end performs two majors tasks: *sampling* and *update* as shown in Figure 4.4. The incoming micro-batches are used to update the user information (i.e., $\mathcal{U}_i$), the item information (i.e., L_i, M_i, Q_i), and the item-pair information (i.e., L_{ij}, M_{ij}, P_{ij}). Next, the back-end employs these updated information along with a biased sampling technique (explained in the following subsection) to compute the item-item similarities on-the-fly (Equation 4.2) and also update the item-item nearest neighbor graph. Lastly, it employs this updated nearest neighbor graph to compute the most up-to-date recommendations for the active users and then forwards these recommendations to the front-end.

A key advantage of this front-end, back-end design is parallelism, separating the two different functionalities of SWIFT, namely *recommendation request handling* (front-end) and *incremental update* (back-end). The information between the front-end and back-end is transferred via the network in a compressed gzip format in order to avoid an additional energy overhead.

This design also provides *flexibility* to our system as the size of the micro-batch can be tuned. The service provider that hosts SWIFT can choose the frequency of the updates depending on the available resources. A small start-up company using SWIFT can aim for a medium-sized micro-batch (say around 100 events per micro-batch) to trade the additional costly updates for relatively less accurate similarity values. By setting a micro-batch size of 1, SWIFT performs stream processing (similar to TENCENTREC [92]). The micro-batch size can also be automatically set by the front-end based on the rate of incoming events as well as the estimated latency of the back-end such that bigger micro-batches can be used at peak usage times. Additionally, the front-end can temporarily increase the micro-batch size to allow for



(a) *Candidate set for an item (in black).*

(b) *Convergence for ML-1M.*

Figure 4.5 – *The biased sampling technique of* SwIFT.

some back-end maintenance. The ability to trade between stream and micro-batch processing of new ratings, depending on the users' demands, highlights the flexibility of our approach.

Biased sampling. Calculating all the similarity pairs for every new update would lead to a prohibitive $\mathcal{O}(|\mathcal{I}|^2 * |\Delta\mathcal{U}|)$ time complexity for each update where $\mathcal{I}$ denotes the set of all items and $\Delta\mathcal{U}$ denotes the set of users who provided new ratings. In the average case, a small fraction of the total similarity pairs is significantly affected after an update. Therefore, updating the similarities only for the aforementioned small fraction of item pairs and using stale values for the rest would notably reduce time complexity without compromising the recommendation accuracy. A sampling method is required for carefully selecting the item pairs to be updated, balancing the trade-off between the number of updates and the recommendation accuracy.

We apply an incremental biased sampling technique (similar to HyRec in §3.1) to address this issue. Our sampling technique is applied in an item-based manner as item-item similarities are more stable than user-user similarities [94]. This biased sampling technique is illustrated in Figure 4.5(a). The black item i is the most recently rated item. Region 1 contains the K-nearest neighbors of i which we will reference to as one-hop neighbors ($\text{kNN}_i^{(1)}$). Region 2 contains K^2 two-hop neighbors of i ($\text{kNN}_i^{(2)}$). Finally, region 3 contains K random items ($Rand(K)$), thus creating the *candidate set*[4] of maximum size: $1 + K + K * K + K = (K+1)^2$ items. The random neighbors are required in order to update the similarities for some items that are not in the two-hop neighborhood. Therefore, the function for selecting the K-nearest neighbors is not stuck at a local minimum. This technique results in a convergence to neighbors of *good* quality[5] within a few updates and eventually converges to the optimal top-K (Figure 4.5(b)).

Theorem 3 (BIASED SAMPLING)**.** *The incremental biased sampling eventually converges to the optimal top-K neighbors.*

Proof. First, we mathematically denote the candidate set at timestep t: $cand_i(t) = \{\text{kNN}_i^{(1)}(t-$

---

[4] The candidate set consists of all the items for which the information (i.e. P, Q, L, M) is incrementally updated by SwIFT's back-end.

[5] Good quality neighbors are the neighbors with relatively high similarity.

$1) \cup \text{KNN}_i^{(2)}(t-1) \cup Rand(k)\}$. Our biased sampling technique results in a directed graph $G_{\text{KNN}}(t)$ that connects each item with a set of items $\text{KNN}_i^{(1)}(t)$ that maximizes the similarity function $S_{ij}(t)$:

$$\text{KNN}_i^{(1)}(t) = \max_{j \in cand_i(t)} \sum_{j=1}^{K} S_{ij}(t)$$

After T iterations, the scanned items consist of $\bigcup_{t=1}^{T} cand_i(t)$. Moreover, we have $\bigcup_{t=1}^{T} cand_i(t) \xrightarrow{T \to \infty} \mathcal{I}$ where $\mathcal{I}$ is the set of all items. Hence, our biased sampling technique eventually converges to the optimal top-K neighbors. □

Figure 4.5(b) depicts the fast convergence of our biased sampling as compared to a random sampling technique where the candidate set does not include the two-hop neighbors ($cand_i(t) = \{\text{KNN}_i^{(1)}(t-1) \cup Rand(k)\}$). The *view similarity* denotes the average similarity of the top-K neighbors at any given update step.

SWIFT's sampling technique improves the incremental update time complexity to $\mathcal{O}((K+1)^2 * |\Delta\mathcal{U}|) = \mathcal{O}(|\Delta\mathcal{U}|)$. Note that there are other sampling techniques used to speedup K-nearest neighbor computation like the one in TENCENTREC with $\mathcal{O}(|\mathcal{I}| * |\Delta\mathcal{U}|)$ time complexity for each incremental update which makes our sampling technique significantly faster.

Recommendation. We implement item-based CF (Algorithm 2) by executing the following phases in SWIFT.

- We substitute the *similarity computation phase* by leveraging our novel I-SIM metric.
- The *neighborhood computation phase* leverages the candidate set selected using our *item-based biased sampling* technique to reduce the time complexity of the K-nearest neighbor search. More precisely, we replace the item set I with the candidate set in the *GetSimilars* function within Phase 2 of Algorithm 1.
- For the *prediction phase*, we apply the prediction score function, shown in Equation 2.4, to generate the final predictions. We reduce the computations by predicting only for the top 10% of the items sorted by popularity. We then compute the top-N recommendations by sorting the prediction scores.

One general problem for a recommender is the cold-start, when recommendations are required for *new items* (i.e. items with no previous ratings in the database). In SWIFT, we initially assign the K most popular items as neighbors for the new item. Neighbors converge to the K-nearest ones after a few iterations for this item as we demonstrate in Figure 4.5(b).

B. I-TRUST: Trust-distrust Predictor in OSNs

To demonstrate the efficiency of I-SIM in trust-distrust predictions (§2.6), we plug I-SIM$_{\epsilon=0}$ in a trust-distrust prediction application which we call I-TRUST.

Temporal behavior also exists in trust-distrust relationship in OSNs. For example, the trust between an elector and voters might change over time. One such behavior is demonstrated in the *Wiki-Elections* trace [185]. We observe a decreasing trend in the number of votes on Wiki-Elections as shown in Figure 4.6. More intuitively, this shows that during the first election, the voters' trust for this wikipedia administrator decreases with time due to more negative votes (distrust).

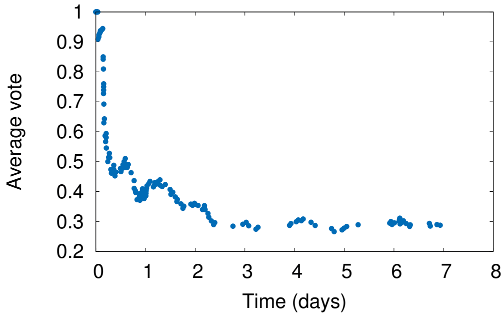


Figure 4.6 – *Voters' trust in an administrator during a Wiki-Election*

We design a trust predictor which captures these temporal effects. We employ Algorithm 3 for two classes (C_0: Trust, C_1: Distrust) to predict the trust relationships. We plug I-SIM$_{\epsilon=0}$ in the similarity computation phase. Based on Equations 4.11 and 4.12, we update the similarity computations incrementally after some given number of events during which $\mathcal{O}(|\Delta U|)$ users were active. The time complexity of each update step then decreases from $\mathcal{O}(|U|)$ to $\mathcal{O}(|\Delta U|)$ as shown in §4.1.2. As we demonstrate later in our experimental evaluation, I-TRUST's incrementality improves the latency significantly whereas its temporality improves the prediction accuracy.

4.1.4 Evaluation

In this section, we report on the performance of our two applications (SWIFT and I-TRUST) in terms of accuracy, latency and energy consumption. Then, we compare them with state-of-the-art alternatives on real-world traces.

A. Experimental Setup

We first describe our experimental environment along with our methodology for obtaining the results.

Platform. We select the *Grid5000* testbed[6] as our experimental platform. Each cluster on Grid5000 has a set of nodes with specific resources. We measure the energy consumption of our implementations using Grid5000's customized Wattmeter which monitors the power consumption.

Unless stated otherwise, we deploy our implementations on a Spark cluster consisting of four nodes. Each node consists of two six-core Intel Xeon E5-2630 v3 CPUs, 128 GB of memory along with 600 GB disk storage. We tune our Spark cluster optimally in order to achieve the best possible performance in terms of the number of partitions and executors per node. We empirically found that the optimal performance, in terms of latency, is obtained by using one executor per machine and setting the number of partitions for all RDDs approximately equal to the total number of physical cores in the Spark cluster.

Datasets. We use publicly available real-world datasets. More specifically, we use *MovieLens* datasets [138]: ML-1M and ML-20M. The ML-1M dataset consists of 1,000,209 ratings from 6040 users on 4000 movies. The ML-20M dataset consists of 20,000,263 ratings from 138,493 users on 27,278 movies. *Rating density* denotes the fraction of actual ratings collected among all possible ratings. To evaluate the effect of increasing the rating density, we use a densified[7] Flixster dataset by employing the method introduced in [122] which leads to 5,105,850 ratings from 10,000 most active users on 4000 most popular movies. Finally, for evaluating I-TRUST we employ the *Wiki-Elections* dataset [185] containing 114,029 votes from 6210 users on 2391 editors.

Metrics. We evaluate both our applications from various aspects. We describe below the metrics used in our evaluation.

Click-Through-Rate (CTR). We adopt this metric to test the accuracy of the recommendations. Given that $\mathcal{H}_u$ is the set of recommended items that were clicked by a user u (hits), and $\mathcal{R}_u$ is the set of items recommended to u, we denote the CTR for u by CTR_u and define it as follows: $CTR_u = |\mathcal{H}_u| / |\mathcal{R}_u|$

The overall CTR over the whole test set is the average over the CTR values for all users in the test set. Note that a recommended item is considered as a *hit*, if the user rates that item anytime later than the time of the recommendation. Ideally, CTR for e-commerce services varies between 1%-5% depending on the type of service [106].

Recall. As introduced in §2.7, this metric captures the sensitivity of a recommender to the frequency of updates. Given that $\mathcal{C}_u$ is the set of items clicked by a user u, we denote the recall for u by $Recall_u$ and define it as follows: $Recall_u = |\mathcal{H}_u| / |\mathcal{C}_u|$. The overall recall is the average over the recall values for all the users in the test set.

Classification accuracy. We use this metric to test the accuracy of trust-distrust predictions in

---

[6] https://www.grid5000.fr/

[7] The density for ML20M is 0.0053, for ML1M 0.045, and for Flixster 0.128.

OSNs. More precisely, the classification accuracy is the fraction of correct predictions among all the predictions.

Mean Absolute Error (MAE). We employ this metric to ensure a fair comparison with model-based alternatives which optimize for low prediction error. The MAE is defined as follows: $MAE = \sum_{u,i \in S} |\hat{r}_{ui} - r_{ui}|/|S|$, where $\hat{r}_{ui}$ denotes the rating prediction for user u and item i, r_{ui} denotes the actual rating and S denotes the set of test rating events. Since MAE captures how close the predictions are to the actual ratings, the lower the error, the higher the model prediction accuracy.

Latency. This metric quantifies the delay observed to complete a single task. This delay consists of three main parts: CPU time, I/O time, and communication delay (e.g. if data is scattered on multiple nodes). For a set of tasks, we show the minimum, median and 99^{th}-*percentile* latency[8].

Energy-per-click. This metric quantifies the amount of energy required for performing computations for a single user click. This metric intuitively evaluates the *impact of a single click* on the consumed energy. More precisely, we measure the aggregated energy consumption of the entire cluster, on which we deploy our experiments, for the operations that a single recommendation task (click) triggers. Given that $\bar{P}$ denotes the average cluster power consumption throughout the computation time of a click (denoted as t), the energy consumption is computed as follows: $E = \bar{P} * t$. We measure the energy-per-click in terms of watt-hour (Wh).

Evaluation scheme. The datasets include the timestamp for each event. We replay the dataset, ordered by the timestamp, to capture the same temporal behavior as the original one. Furthermore, we split the dataset into *training*, *validation* and *test* sets. Based on the benchmark for evaluating stream-based recommenders [105], our test set consists of the most recent 1000 ratings. The validation set consists of the last 1000 ratings from the training set and is used for parameter tuning. For the non-incremental competitors we train the model on the training set until it converges and then we evaluate the trained model on the test set.

B. SWIFT Evaluation

SWIFT is designed to provide accurate recommendations with low latency in an energy-efficient manner. In this section, we evaluate SWIFT's performance for varying parameter settings and then compare it with state-of-the-art incremental and non-incremental competitors.

To compare with incremental recommenders, we consider TENCENTREC's practical item-based CF (which we refer to as TENCENTREC). Compared to SWIFT, TENCENTREC's practical algorithm employs incremental approximate cosine similarity (instead of I-SIM) with *real-time*

---

[8] The latency observed by 99% of the tasks is below this value.

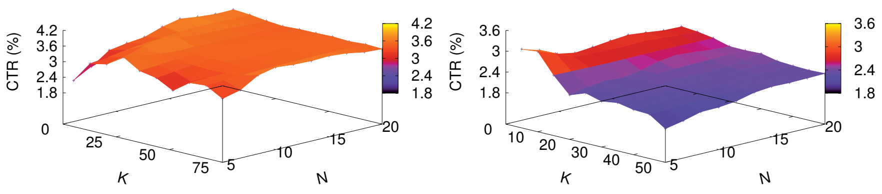
pruning (instead of biased sampling) and *real-time personalized filtering* while predicting only for the top 10% of the items sorted by popularity similar to SWIFT (Phase 3 in Algorithm 2).

For the non-incremental alternatives, we compare with a standard matrix factorization based recommender using temporal relevance (TIMESVD [110]) as well as with the factored similarity models (FISM [99]), both of which are publicly available in the LIBREC[9] library for recommenders. Additionally, we compare with the distributed alternating least squares (ALS) algorithm available in Spark's MLlib.

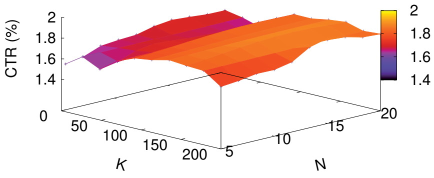
We train SWIFT using the training set and then provide recommendations for each rating event in the test set. More precisely, for the training set, SWIFT computes the required information (P, Q, L, M) based on the Equations 4.3, 4.4, 4.5, 4.6 of the adjusted-cosine similarity (Equation 4.2). For the test set, SWIFT updates this information using Equations 4.7, 4.8, 4.9, 4.10 and then provides recommendations using the updated information. Depending on the flexibility mode, the back-end is invoked for the update operations either per click (stream processing) or per micro-batch (batch processing). In the stream processing mode, the front-end responds to the clients' requests only after receiving the updated recommendations from the back-end.

Accuracy. The following experiments demonstrate the effect of SWIFT's parameters on the recommendation accuracy, namely: *model size* (K), *recommendations-per-click* (N), *micro-batch size* (L) and *temporal relevance* (α).

Model size. We measure the CTR while varying the model size (K) which is the number of neighbors in the item-item network. We observe in Figure 4.7 that after a certain model size any further increase in the model size reduces the CTR. This decrease in CTR is due to the inclusion of less similar neighbors in the neighborhood of an item. These less similar neighbors add noise to the predictions.



(a) *ML-1M* **(b)** *ML-20M*



(c) *Flixster*

Figure 4.7 – *Impact of model size (K) and recommendations-per-click (N) on accuracy.*

---

[9] http://www.librec.net/

Recommendations-per-click. The number of recommendations provided per click, is another important parameter that affects the CTR as too few will be insufficient whereas too many will reduce the interest of users in the recommendations. Hence, it is important to highlight that in practical recommenders, the recommendations-per-click (N) should not exceed 20. For example, IMDB uses Top-12 list to suggest movies and Last.fm uses Top-5 list to suggest songs. We observe a steady behavior in CTR with increasing N as shown in Figure 4.7. This behavior can be attributed to the fact that the size of the recommendation hits grows proportionally to the size of the recommended items.

Micro-batch size. Recall that SWIFT provides a flexible back-end as mentioned in §4.1.3. More precisely, SWIFT provides recommendations treating each stream of rating events as a micro-batch. Hence, SWIFT can provide stream processing with the micro-batch size set to 1 whereas the micro-batch size can be set to few hundreds of rating events for batch processing. Note that this flexibility is an important feature for practical recommenders, as depending on the available resources (due to limited operational costs) or the network traffic (due to multiple recommendation requests), the micro-batch size can be adjusted by the service provider hosting SWIFT.

We now evaluate the impact of the flexibility mode on accuracy. Practically, many recommenders like Amazon or eBay repeat certain recommendations similar to SWIFT. Such repeated recommendations are less frequent in the stream processing mode (more frequent updates in top-N recommendations) but occur more often as the micro-batch size increases. Therefore, the denominator of the CTR (number of recommended items) decreases as the micro-batch size increases. On the contrary, the denominator of the recall (number of clicked items) is independent of the micro-batch size. More updated recommendations (smaller micro-batch size) lead to more hits and thus result in an increase in the numerator. Hence, we employ the recall to capture the difference in accuracy for varying micro-batch sizes.[10]

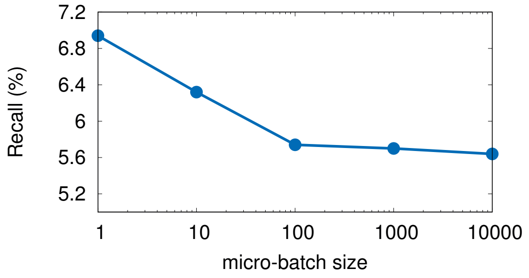


Figure 4.8 – *Impact of flexibility mode on accuracy for ML-1M.*

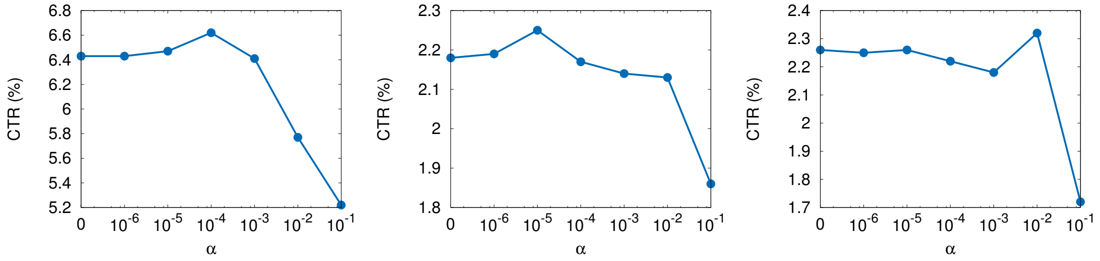
More precisely, Figure 4.8 illustrates this trade-off between accuracy and micro-batch size. Compared to the stream processing mode (micro-batch size set to 1), there is an impact on the recommendation accuracy, in terms of recall, for the batch processing mode. Furthermore, there is a steep decrease in the recall with increasing micro-batch size. This behavior is due to

---

[10] Note that all the experiments leveraging the CTR metric have a fixed micro-batch size.

less frequent updates leading to more temporally stale similarities.

Temporal relevance. We analyze the effect of temporal relevance on the quality of recommendations in terms of CTR. For these experiments, we increase the test set to the last 10,000 events as the drift in the users' interests is more evident over longer test periods. We set the micro-batch size to 100 and tune the degree of temporal relevance by regulating the temporal weight parameter α. We observe an improvement in the CTR while increasing the value of α as shown in Figure 4.9. Moreover, we also observe that the CTR starts decreasing at some point. This outcome occurs due to the fact that many of the users rated very few items and our item-based approach leverages the items in the profile of the user. Hence, an increased value of α results in degrading the already few ratings in the user profile leading to a cold-start scenario for the given user. Note that we can also vary α specifically for each user profile; this is left for future work.



(a) *ML-1M* **(b)** *ML-20M* **(c)** *Flixster*

Figure 4.9 – *Impact of temporal relevance (α) on accuracy. Setting α to 0 deactivates* SWIFT*'s temporal feature.*

Table 4.1 compares SWIFT with incremental recommenders (TENCENTREC) as well as with non-incremental ones (TIMESVD, ALS, FISM) in terms of mean absolute error in predictions. We observe that SWIFT outperforms the others on the more sparse datasets (ML-1M, ML-20M) whereas ALS performs best on a relatively dense dataset (Flixster).

Dataset / Approach	*ML-1M*	*ML-20M*	*Flixster*
FISM	0.731	0.873	0.713
TIMESVD	0.806	0.892	0.73
ALS	0.707	0.746	**0.629**
SWIFT	**0.686**	**0.662**	0.669
TENCENTREC	0.784	0.721	0.684

Table 4.1 – *Model comparison (MAE) between incremental and non-incremental alternatives.*

Latency. SWIFT's latency is primarily affected by the *model size* (K), *micro-batch size* (L) and *cluster size* parameters. We now provide the results concerning SWIFT's latency for different settings for these parameters.

Model size. SWIFT's biased sampling depends on the model size (K). An increase in the model size generates larger candidate sets ($\mathcal{O}(K^2)$ size) thereby leading to more computations. Figure 4.10 depicts that the increase in the computations is more evident for large and sparse datasets like ML-20M. This behavior is due to the fact that the larger amount of items in the database combined with the sparsity leads to more diverse items in a candidate set. Hence, the amortized complexity of our biased sampling increases. In this specific case, the biased sampling does not reduce the computations with large values of K, thereby having a significant impact on latency (as shown in Figure 4.10 for ML-20M and $K = 200$).

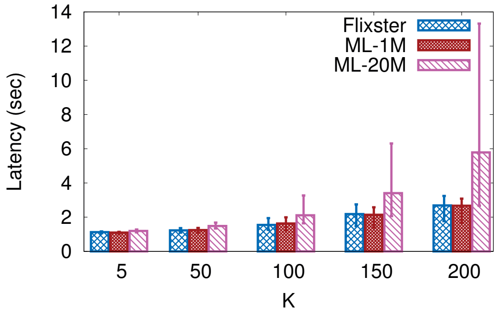


Figure 4.10 – *Impact of model size (K) on latency (stream processing).*

Micro-batch size. We evaluate the flexibility of SWIFT by varying the micro-batch size. Figure 4.11 shows the recommendation and update latency of SWIFT's front-end and back-end respectively for $K = 50$. The update latency is increasing with the micro-batch size as the information for more items' candidate sets needs to be updated. Nevertheless the recommendation time is nearly the same for varying micro-batch size. The latency observed between a click and the generation of the recommendations is a few milliseconds. Note that in the batch processing mode, the similarities are updated only after the system receives a micro-batch of L fresh ratings.

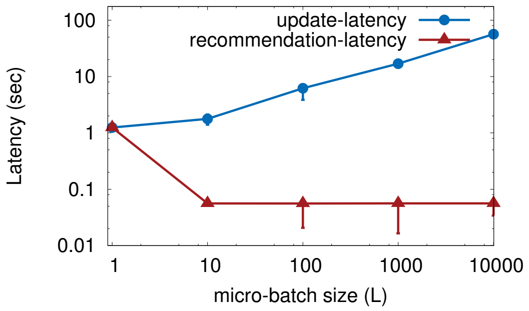


Figure 4.11 – *Impact of batch processing on latency for ML-1M.*

Cluster size. We deploy SWIFT and ALS on the same cluster while increasing the cluster size (number of nodes in the cluster) and compare the improvement in terms of median latency (which we quantify as speedup). Figure 4.12 demonstrates that SWIFT (stream processing mode with the model size set to 200) achieves a better speedup than ALS. Furthermore, an increase in the micro-batch size leads to an increase in the speedup for SWIFT. Therefore, the increase in the update latency, shown in Figure 4.11, can be mitigated by employing more nodes due to SWIFT's scalability.

The scalability saturates after a certain cluster size (5 nodes) due to the communication time with Cassandra as well as the sequential dependencies among SWIFT's tasks. The communication overhead with Cassandra could be possibly mitigated by using a distributed Cassandra cluster and tuning it to maximize the benefits from locality whereas the sequential dependencies could be reduced by pipelining the tasks to exploit more parallelism. It is important to note that the observed bottleneck is implementation specific and not a limitation of I-SIM.

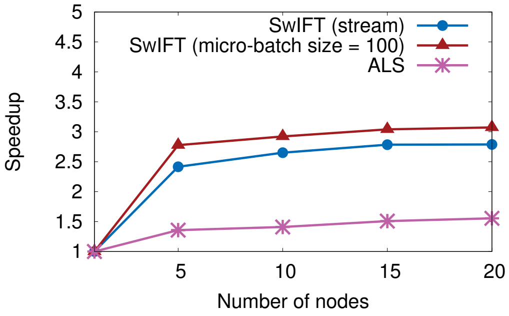


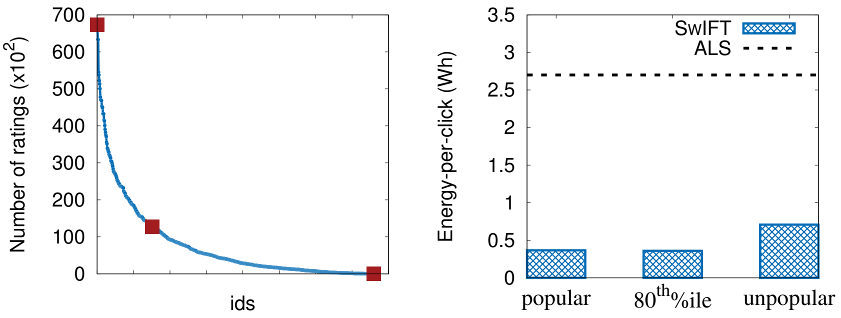
Figure 4.12 – *Scalability comparison for ML-20M.*

Energy Consumption. We evaluate the energy consumed by the computations induced due to a user click. In other words, we estimate the impact of a single click on energy consumption. Recall that our goal is to reduce the energy consumption by reducing the time complexity. We analyze the energy consumption corresponding to the clicks for three representative items: *most popular*, *least popular* and *80th percentile*[11]. The ratings provided by users follow a long tail distribution (Figure 4.13(a)) where 80% of the users rate only 20% of the items. Hence, we choose our 80th percentile item along with the most popular and unpopular items as shown in Figure 4.13(a).

Figure 4.13(b) depicts the energy consumption of SWIFT ($K = 100$) for clicks corresponding to these three items. The unpopular items are not strongly correlated to their neighbors due to the relatively small number of ratings provided for each of them. Therefore, the items in their candidate sets have less overlap compared to those in the candidate sets of the more popular items. Thus, there is an increase in the computation time for the unpopular items leading to an increase in the energy-per-click. We deploy ALS on the same Spark cluster for benchmarking the energy consumption of a single update on this cluster (Figure 4.13(b)). Note that ALS is non-incremental and therefore requires significantly more time for one update than SWIFT, thus leading to higher energy consumption.

---

[11] The 80th percentile popular item is the one with popularity higher than 80% of the items.



(a) *Item Popularity* **(b)** *Energy-per-click*

Figure 4.13 – *Impact of item popularity on energy consumption for ML-20M.*

C. I-TRUST Evaluation

We now evaluate the effectiveness of I-TRUST in providing accurate predictions with low latency. We denote the classical predictor implementing Algorithm 3 as C-TRUST. For the experiments, we set the model size (K) to 150 for C-TRUST to achieve the optimal quality. We have the same model size with the temporal parameter (α) as 0.3 for I-TRUST. We deploy these experiments on a single node. While training I-TRUST, we update the similarities incrementally after a fixed micro-batch of training events whereas for C-TRUST the similarities are computed using all the training events in a non-incremental manner.

Runtime. We measure the total runtime for updating the similarities needed for constructing the K-nearest neighbor graph using all the training events. This graph is then used to predict the trust relations as shown in Algorithm 3 (Phase 3). For I-TRUST, we set the micro-batch update for similarity computations to 1000 voting events. From Table 4.2, we observe that the runtime improves by 36 times.

Accuracy. Table 4.2 confirms I-TRUST's superiority in terms of accuracy. I-SIM$_{\epsilon=0}$ incorporates the time-varying trust relations between an administrator and the voters, in the similarity values. Therefore, the k-nearest neighbor graph is temporally more accurate and leads to better predictions. The improvement is reflected in the difference with C-TRUST for the voting classification task.

Approach	Runtime	Classification Accuracy
C-TRUST	421.2 s	79.21%
I-TRUST	**11.66 s**	**80.75%**

Table 4.2 – *Runtime and accuracy comparisons for* I-TRUST *and* C-TRUST.

4.1.5 Related Work

Collaborative filtering. CF algorithms can be generally divided into two categories: *memory-based* and *model-based*. Memory-based algorithms employ user-item ratings to compute the predictions and then generate relevant recommendations. These algorithms can be either *user-based* [83] or *item-based* [157]. Our work focuses on the item-based CF technique which has been shown to provide more accurate recommendations compared to the user-based one [157]. In contrast to memory-based techniques, model-based ones build parametric models by learning iteratively on the training datasets and then leverage the learned model to generate predictions. Different types of models are typically used, including matrix factorization [110] and factored item similarity models [99]. Standard model-based techniques require to update their learned models by employing all the ratings, including the new ones, and hence are not incremental in nature.

Real-time recommenders. These have recently attracted a lot of attention. Huang et al. presented TENCENTREC, a real-time stream recommender [92] which uses an incremental version of approximate cosine similarity. We demonstrate in §4.1.4 that by trading storage (to store the L and M information), I-SIM performs better in terms of accuracy compared to the similarity metric leveraged by TENCENTREC. Furthermore, SWIFT's biased sampling is significantly faster than TENCENTREC's real-time pruning as we explained in Section 4.1.3. Whilst Yang et al. [187] presented a scalable item-based CF method by using incremental update, they did not however address the problem of temporal relevance.

Temporal relevance. Few approaches have addressed the problem of temporal relevance in the context of CF. One simple heuristic to capture the temporal behavior of a user, applicable to any recommender, is to consider only the most recent ratings in her profile for generating the recommendations [92, 32, 37]. In our work, we focus on the temporal relevance in the context of similarity computations. Ding et al. [53] exploited the timestamps of ratings to adapt the item-based CF technique. They incorporated time-based weights in the score prediction stage but did not adapt the similarity computations, hence leading to higher time complexity. Lathia et al. [113] analyzed the effect of temporal relevance by varying the neighborhood size over time. Koren et al. [110] designed a matrix factorization model that considers the temporal behavior of users. However, their model has a higher time complexity as they employ multiple time dependent parameters. Liu et al. [122] introduced an incremental version of cosine similarity that provides temporal relevance. However, Sarwar et al. [157] empirically showed that an item-based CF technique provides more accurate recommendations by leveraging the adjusted cosine metric (compared to the classical cosine one). I-SIM provides incremental updates for the adjusted cosine similarity while incorporating the temporal relevance feature.

Energy-efficiency. Despite a large amount of work on large-scale CF [198, 188, 157], none of the existing approaches focuses on reducing the time complexity. The main focus has been so far to design distributed algorithms which can decentralize the computations over multiple nodes leading to better scalability. This strategy leads to more resource utilization and thereby higher energy requirements. However, energy consumption is currently a major concern in data centers [109]. Energy costs are quickly rising in large-scale data centers and are soon projected to overtake the cost of hardware. Energy-efficiency is the new holy grail of data management systems research [81]. We address this energy-efficiency issue by designing incremental computations with lower time complexity.

Trust-distrust in OSNs. Trust inference algorithms rely on users' feedback to predict future trust relations. However, trust relations are assumed to be static in existing literature [126, 197]. In this work, we first demonstrate that trust relations can be time-varying and then present how to capture these dynamic trust relations by leveraging I-SIM and thus enabling lightweight incremental similarity updates.

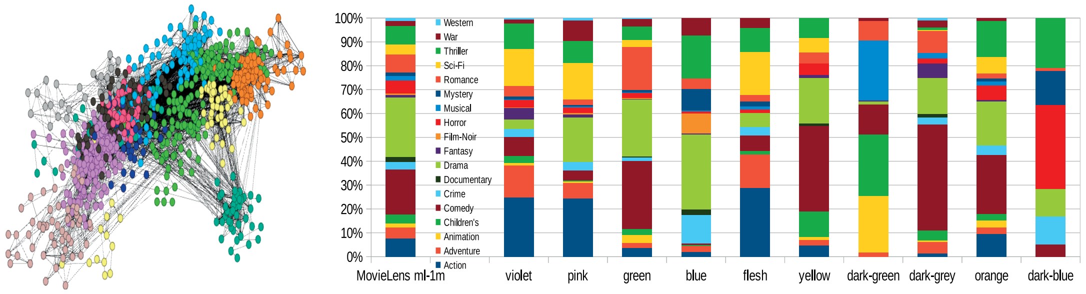
4.1.6 Conclusion

We present I-SIM, a novel similarity metric that enables similarity computations in an incremental and temporal manner. We illustrate through two applications the effectiveness of I-SIM in practice: (a) SWIFT incorporating I-SIM for recommendation and (b) I-TRUST incorporating I-SIM$_{\epsilon=0}$ for trust prediction. We empirically show that I-SIM leads to better accuracy and lower latency along with energy efficiency compared to state-of-the-art alternatives. Moreover, I-SIM can be leveraged to incorporate time-awareness in similarity-based applications, for example, trust recommendation in mobile ad-hoc networks [126] or predictive blacklisting against malicious traffic on the Internet [171].

4.2 CIP: Consumed Item Packs

4.2.1 Overview

In §4.1, we observe how we can design recommender systems which incorporate explicit feedback (e.g., ratings) in an incremental manner while preserving their temporality. Yet, relying on explicit feedback raises issues regarding feedback *sparsity* (in systems where the item catalog is large, users tend to give feedback on a trace amount of those items, impacting the quality of recommendations [2]), and limited efficiency for recommending fresh items in reaction to recent user actions [122]. In this work, we investigate the existence of a higher level abstraction for sequences of consumed items, and algorithms for dealing with them. Our *Consumed Item Packs* (CIPS) relate to high order relations between items enjoyed by a user and therefore eliminating the need of explicit feedback. Some previous works such as HOSLIM [39], considered the consumption of items by the same user as the basis for implicit recommendation. HOSLIM places the so called *user-itemsets* (implicit feedback) in a matrix, and then computes the similarity of jointly consumed items over the whole user history (that leads to the optimal recommendation quality). High-order relations are sought in principle, but due to the tractability issue of this approach (for m items and order k: $\mathcal{O}(m^k)$ combinations of the items are enumerated and tested for relevance), authors limit computations only to pairs of items. Very recently, Barkan et al. proposed to consider item-item relations using the model of word embeddings [16]. Our work generalizes the notion of implicit item relations, based on consumption patterns.



(a) *Communities of movies (MovieLens).* **(b)** *Distribution of genres in the 10 largest communities of the movie graph. (Legend-colors on the x-axis correspond to colors of communities.*

Figure 4.14 – *Existence of temporal consumption habits of users in MovieLens dataset.*

To get more intuition about the very notion of *consumed item packs*, consider the following experiment we conduct on the publicly available MovieLens 1M dataset, from which we extract an undirected graph. Vertices of the graph are movies. *An edge exists between two movies if some minimal number (M) of users have consumed both of them in a "short" consumption interval* (here "short" means consumed within -2 to 3 contiguous hops in the users' consumption log).[12]

In the graph presented in Figure 4.14(a), we only depict, from the original graph, movies where the edges have at least 30 transitions (*i.e.*, 30 users have consumed the two movies within the specified consumption interval, leading to the representation of 1% of the total number of edges). The edges of the graph are weighted by the number of transitions, which is then at least 30 ($M = 30$).

We then apply a community detection algorithm [23] to the resulting graph. We use *modularity* as a measure of the structure of the network. The value of the modularity [23] lies in the range [-1,1]. It is positive if the number of edges within groups exceeds the number expected on the basis of chance. For a given division of the network's vertices into some modules, modularity reflects the concentration of edges within modules compared with random distribution of links between all nodes regardless of modules. A high modularity score (0.569) indicates the presence of strong communities in the graph presented in Figure 4.14(a). We highlight communities which represent at least 1% of the total number of nodes in the original graph. There are 10 such communities, each ranging from 1.08% to 5.21% of the original graph nodes. The average clustering coefficient of the graph is 0.475, the one of the largest community (in purple) is 0.771, and the one of the smallest community (in dark blue) is 0.842. Thus, community clustering is significantly more important than the graph one (which supports the observed high graph modularity). Interestingly, those communities are then (densely) connected, by a *latent feature*.

It is important to notice that this latent feature cannot be reduced to the *genre* of the movies. To show this, we also plot the distribution of movie genres in the 10 (strong) communities

---

[12] The +/- signs denote the order of consumption for the pair of movies.

in Figure 4.14(b). We first observe that each community conveys a very specific blend of genres: one community cannot be trivially reduced to a genre. Secondly, it appears that some communities are closer than others: "pink" and "orange" communities are well separated, both by hop-distance on the graph (Figure 4.14(a)) and by their constituent genres (Figure 4.14(b)). The latent feature cannot be reduced to item *launch times* either: *e.g.*, movie launch times of the smallest of the 10 clusters spread from 1931 to 1997.

We conduct a similar experiment for a product review website (Ciao [40]), setting $M = 2$ on this very sparse dataset. The resulting weighted graph, with detected item communities, also has a high modularity score of 0.61.

In short, these experiments highlight the very existence of a non trivial latent feature, namely consumed item packs (CIPS), somehow representing the temporal consumption habits of users. Extracting this latent information from item communities and then using it for personalization services is not straightforward.

4.2.2 Consumed Item Packs (CIPS)

To get access to this latent feature from service logs, we define the CIP data structure. CIPS are extracted from users' consumption patterns, and allow us to compute the similarity between those users (or items consumed by them). A user's profile is composed of multiple CIPS. The notion of CIP is then instantiated in three different algorithms: a user-based algorithm, an item-based one, and a word embedding based one.

To make things more precise, we recall from §2 our notations: a set of m users $\mathcal{U} = \{u_1, u_2, ..., u_m\}$ and a set of n product catalog items $\mathcal{I} = \{i_1, i_2, ..., i_n\}$. The profile of a user u, noted P_u, consists of a set of pairs of the form $\langle i, t_{ui} \rangle$ (where u consumed an item i at a time $t_{u,i}$), extracted from service logs. CIPS are composed of items: each CIP $\in \mathcal{I}^*$. The order of the items in a given user's CIP represents their relative appearance in time, the leftmost symbol being the oldest one:

$$\mathrm{CIP}_u = [i_1, i_2, i_3, ..., i_k] \ such\ that\ t_{u,i_1} < t_{u,i_2} < ... < t_{u,i_k}.$$

For instance, u_1's CIP (CIP_1) is $[i_{14}, i_3, i_{20}, i_{99}, i_{53}, i_{10}, i_{25}]$, while u_2's one (CIP_2) is $[i_{20}, i_{53}, i_4]$. Items i_{14} and i_{25} are respectively the first and last items that u_1 has consumed in CIP_1, while i_{20} and i_{53} are two items that both users have consumed. In the rest of the work, we assume that one item occurs only once in a given CIP.[13]

A CIP then represents the items consumed by a user over a predefined period of time. Using such a data structure, one can devise a *similarity* measure $sim : \{\mathcal{I}^* \times \mathcal{I}^* \to \mathbb{R}^+\}$[14] between two CIPS, that captures the proximity between users (or items) as we explain in the next two

---

[13] Our similarity metrics might be extended to take re-consumption into account, but it is outside the scope of this work.

[14] $\mathcal{I}^*$ refers to the set of finite length sequences of items from $\mathcal{I}$.

sections.

In practice, CIPS are directly derived from service platform transaction logs, that are at least composed of tuples of item-id and the corresponding consumption *timestamp*[15] of that item. (It is important to note that an explicit recommender system requires tuples including, *in addition*, the rating (r_{ui}) that u provided for item i.)

4.2.3 CIP Algorithms

The core claim of this work is that the notion of CIP is general enough to capture different algorithms that rely on sequences of items. In the next three subsections, we present novel algorithms that determine CIP-based similarities and leverage sequence of items for recommendations.

A. CIP-U: User-based Recommender

In this subsection, we introduce our user-based algorithm using CIPS, which we denote CIP-U. We then present how to perform incremental updates with CIP-U.

CIP-U Algorithm. CIP-U is an incremental algorithm that maintains a user-user network where each user is connected to the most similar K other users. CIP-U exploits users' CIPS, and accepts batches of items freshly consumed by users (*i.e.*, last logged transactions on the platform) to update this network.

P_u^l denotes the profile of a user u till the l^{th} update of her consumed items while CIP_u^{l+1} denotes the batch of new items consumed by her since the last batch update. Assuming $P_u^l = i_1 i_2 ... i_k$ and $\text{CIP}_u^{l+1} = i_{k+1} i_{k+2} ... i_n$, we can denote the profile of a user u after the $(l+1)^{th}$ iteration as $P_u^{l+1} = P_u^l \cup \text{CIP}_u^{l+1}$. Note that $\cup$ is an order preserving union here.

Before we provide the similarity measure to compare users, we introduce some preliminary definitions. We first introduce the notion of *hammock distance* between a pair of items in the profile of a given user u.

Definition 3 (HAMMOCK DISTANCE)**.** *The hammock distance between a pair of items (i, j) in P_u, denoted by $\mathcal{H}_u(i, j)$, is the number of hops between them.*

For instance, in $P_u = [i_{14}, i_3, i_{20}, i_{99}, i_{53}, i_{10}, i_{25}]$, $\mathcal{H}_u(i_{14}, i_{99}) = 3$.

Based on the hammock distance, we define a *hammock pair* ($\mathcal{HP}$) between two users, as a pair of items that both users have in common.

Definition 4 (HAMMOCK PAIRS)**.** *Given two users u and v, their hammock pairs $\mathcal{HP}_{u,v}$ are the set of distinct item pairs both present in P_u and in P_v, under the constraint that the number of*

---

[15] The timestamp denotes the actual consumption time of the item (in the UNIX format).

hops between the item pairs is at most δ_H.

$$\mathcal{HP}_{u,v} = \{(i,j) \mid \mathcal{H}_u(i,j) \leq \delta_H \;\wedge\; \mathcal{H}_v(i,j) \leq \delta_H \;\wedge\; i \neq j\}$$

Hyper-parameter δ_H denotes the *hammock threshold* and serves the purpose of tuning the CIP-based latent feature considered between related items.

Let $[\,]$ denote the Iverson bracket:

$$[P] = \begin{cases} 1 & \text{if the predicate } P \text{ is True} \\ 0 & \text{otherwise.} \end{cases}$$

Finally, from hammock pairs, we derive the similarity of two users with regards to their CIPs as follows.

Definition 5 (SIMILARITY MEASURE FOR USER-BASED CIP)**.** *The similarity between two users u and v is defined as a function of the cardinality of the set of hammock pairs between them:*

$$sim_{\text{CIP-U}}(u,v) = 1 - (1 - [P_u = P_v]) \cdot e^{-|\mathcal{HP}_{u,v}|} \tag{4.13}$$

We obtain $sim_{\text{CIP-U}} \in [0,1]$, with the boundary conditions, $sim_{\text{CIP-U}} = 0$ if the two users have no pair in common ($|\mathcal{HP}_{u,v}| = 0$ and $[P_u = P_v] = 0$), while $sim_{\text{CIP-U}} = 1$ if their CIPs are identical ($[P_u = P_v] = 1$).

Incremental updates. CIP-U enables incremental updates, in order to conveniently reflect the latest users' consumption in recommendations without requiring a prohibitive computation time. CIP-U processes batches of events (consumed items) at regular intervals and updates the similarity measure for pairs of users. $C_{u,v}$ denotes the set of items common in the profiles of two users u and v. More precisely, after the l^{th} iteration, we obtain:

$$C^l_{u,v} = P^l_u \cap P^l_v$$

Then, at the $(l+1)^{th}$ iteration, we get:

$$\begin{aligned} C^{l+1}_{u,v} &= P^{l+1}_u \cap P^{l+1}_v = (P^l_u \cup \text{CIP}^{l+1}_u) \cap (P^l_v \cup \text{CIP}^{l+1}_v) \\ &= (P^l_u \cap P^l_v) \cup (P^l_u \cap \text{CIP}^{l+1}_v) \cup (P^l_v \cap \text{CIP}^{l+1}_u) \cup (\text{CIP}^{l+1}_u \cap \text{CIP}^{l+1}_v) = C^l_{u,v} \cup \Delta C^{l+1}_{u,v} \end{aligned}$$

where $\Delta C^{l+1}_{u,v}$=$(P^l_u \cap \text{CIP}^{l+1}_v) \cup (P^l_v \cap \text{CIP}^{l+1}_u) \cup (\text{CIP}^{l+1}_u \cap \text{CIP}^{l+1}_v)$. Note that the time complexity of this step is $O((|P^l_u| + |\text{CIP}^{l+1}_v|) + (|P^l_v| + |\text{CIP}^{l+1}_u|))$, where $|\text{CIP}^{l+1}_u|$, $|\text{CIP}^{l+1}_v|$ are bounded by the number of events after which the batch update will take place, say Q. Hence, the time complexity is $O(n+Q) = O(n)$, where n denotes the total number of items, and when $Q << n$ (as expected in a system built for incremental computation).

We next incrementally compute the new hammock pairs. $\Delta\mathcal{HP}_{u,v}$ denotes the set of new hammock pairs for users u and v. Computation is performed as follows:

$$\Delta\mathcal{HP}_{u,v} = \{(i,j) \mid (i \in C^{l}_{u,v}, j \in \Delta C^{l+1}_{u,v}) \ \wedge \ (i \in \Delta C^{l+1}_{u,v}, j \in \Delta C^{l+1}_{u,v}) \ \wedge \ \mathcal{H}_u(i,j) \leq \delta_H \ \wedge \ \mathcal{H}_v(i,j) \leq \delta_H\}$$

The time complexity of this step is $O(|C^{l}_{u,v}| \cdot |\Delta C^{l+1}_{u,v}|)$, where $|\Delta C^{l+1}_{u,v}|$ is bounded by the number of events after which the batch update takes place (Q). Hence, the time complexity is also of $O(n \cdot Q) = O(n)$.

Finally, the similarities are computed leveraging the cardinality of the recently computed incremental hammock pairs. More precisely, we compute the updated similarity on-the-fly between a pair of users u and v after the $(l+1)^{th}$ iteration as follows:

$$sim^{l+1}_{u,v} = 1 - (1 - [P^{l+1}_u = P^{l+1}_v]) \cdot e^{-|\mathcal{HP}^{l}_{u,v} + \Delta\mathcal{HP}_{u,v}|}$$

Hence, the similarity between one user and all m others is computed with a $O(nm)$ time complexity.[16] In CIP-U, we retain only a small number (K) of similar users. For each user u, we retain the K most similar users, where $K << m$, and record these user-ids along with their similarities with u. We term K as the *model size*. Selecting the top-K similar users for collaborative filtering based on their similarity requires sorting, which induces an additional complexity of $O(m \log m)$. Hence, the total time complexity is $O(nm) + O(m \log m) = O(nm)$ (since $n >> \log m$). Note that classical explicit collaborative filtering algorithms like user-based [154] or item-based [157] ones also have same time complexity for periodically updating their recommendation models. We can reduce the time complexity for the top-K neighbors update further to $O(n)$ by using biased sampling and iteratively updating the neighbors [27].

B. CIP-I: Item-based Recommender

In this subsection, we introduce our item-based algorithm using CIPS, which we denote as CIP-I. We then present how to perform incremental updates with CIP-I.

CIP-I Algorithm. CIP-I is also an incremental algorithm that processes user consumption events in CIPS, to update its item-item network.

Similar to CIP-U, we also leverage the notion of user *profiles*: a profile of a user u is noted P_u, and is composed of one or more disjoint CIPS. We use multiple CIPS in a user profile to model her consumption pattern. CIPS are separated based on the timestamps associated with the consumed items: two consecutive CIPS are disjoint if the former's last and latter's first items are separated in time by a given interval (noted δ).

Definition 6 (CIP PARTITIONS IN A USER PROFILE). *Let i_k and i_{k+1} denote two consecutive*

---

[16] Our time complexity analysis concerns the training phase of the recommender as this phase requires more computational effort.

consumption events of a user u, with consumption timestamps t_{u,i_k} *and* $t_{u,i_{k+1}}$*, such that* $t_{u,i_k} \leq t_{u,i_{k+1}}$*. Given* i_k *belongs to* CIP_u^l*, item* i_{k+1} *is added to* CIP_u^l *if* $t_{u,i_{k+1}} \leq t_{u,i_k} + \delta$*. Otherwise* i_{k+1} *is added as the first element in a new* CIP_u^{l+1}*.*

These CIPs are defined as δ-distant. The rationale behind the creation of user profiles composed of CIPs is that each CIP is intended to capture the semantic taste of a user within a consistent consumption period.

With $i <_{\text{CIP}} j$ denoting the prior occurrence of i before j in a given CIP, and the inverse hammock distance ($\epsilon_u(i,j)$) being a penalty function for distant items in a CIP_u (*e.g.*, $\epsilon_u(i,j) = \frac{1}{\mathcal{H}_u(i,j)}$), we express a similarity measure for items, based on those partitioned user profiles, as follows.

Definition 7 (SIMILARITY MEASURE FOR ITEM-BASED CIP)**.** *Given a pair of items* (i,j)*, their similarity (*$sim_{\text{CIP-I}}(i,j) = s$*) is:*

$$s = \frac{\sum_u \sum_{l=1}^{|l|_u} \left[(i,j) \in \text{CIP}_u^l \wedge i <_{\text{CIP}} j\right]\left(1 + \epsilon_u(i,j)\right)}{2 \cdot \max\left\{\sum_u \sum_{l=1}^{|l|_u} \left[i \in \text{CIP}_u^l\right], \sum_u \sum_{l=1}^{|l|_u} \left[j \in \text{CIP}_u^l\right]\right\}} = \frac{score_{\text{CIP-I}}(i,j)}{2 \cdot max\left\{cardV(i), cardV(j)\right\}} \tag{4.14}$$

where $|l|_u$ *denotes the number of* CIPs *in the profile of user u and* [ ] *denotes the Iverson bracket.*

This reflects the number of close and ordered co-occurrences of items i and j over the total number of occurrences of both items independently: $sim_{\text{CIP-I}}(i,j) = 1$ if each appearance of i is immediately followed by j in the current CIP. Contrarily, $sim_{\text{CIP-I}}(i,j) = 0$ if there is no co-occurrence of those items in any CIP. Furthermore, we denote the numerator term as $score_{\text{CIP-I}}(i,j)$ and the denominator term as a function of $cardV(i)$ and $cardV(j)$ subterms for Equation 4.14 where $cardV(i) = \sum_u \sum_{l=1}^{|l|_u} [i \in \text{CIP}_u^l]$. As shown in Algorithm 4, we can update $score_{\text{CIP-I}}(i,j)$ and $cardV(i)$ terms incrementally. Finally, we can compute the similarity on-the-fly leveraging $score_{\text{CIP-I}}(i,j)$ and $cardV(i)$ terms.

Incremental updates. CIP-I processes users' recent CIPs scanned from users' consumption logs. Score values ($score_{\text{CIP-I}}$) are updated as shown in Algorithm 4. We require an item-item matrix to maintain the *score* values, as well as an n-dimensional vector that maintains the current occurrence number of each item.

After the update of the *score* values, the algorithm terminates by updating a data structure containing the top-K closest items for each given item, leveraging the *score* matrix and the cardinality terms for computing similarities on-the-fly.

The complexity of Algorithm 4 depends on the maximum tolerated size of incoming CIPs. As one expects an incremental algorithm to receive relatively small inputs as compared to the total dataset size, the final complexity is compatible with online computation: *e.g.*, if the largest CIP allowed has cardinality $|\text{CIP}| = O(\log n)$, then run-time complexity is poly-logarithmic.

Algorithm 4 *Incremental Updates for Item Pairs.*

```
Require: $\mathrm{CIP}_u$                          ▷ last $\delta$-distant CIP received for user $u$
$score_{\text{CIP-I}}[\,][\,]$                     ▷ item-item score matrix, intialized to 0
$cardV$                                            ▷ $n$-dim. vector of appearance cardinality of items
for item $i$ in $\mathrm{CIP}_u$ do
    $cardV(i) = cardV(i) + 1$
    for item $j$ in $\mathrm{CIP}_u$ do
        if $i \neq j$ then
            $\epsilon(i,j) = \epsilon(j,i) = \frac{1}{\mathcal{H}_u(i,j)}$
        end if
        if $i <_{\text{CIP}} j$ then
            $score_{\text{CIP-I}}$[i][j]+=$(1 + \epsilon(i,j))$
        else
            $score_{\text{CIP-I}}$[j][i]+=$(1 + \epsilon(j,i))$
        end if
    end for
end for
```

C. DeepCIP: Embedding-Based Recommender

In this subsection, we present an approach based on machine learning, inspired by Word2Vec[135, 16]. This approach relies on word embedding, transposed to items. We specifically adapt this concept to our CIP data structure. We name this CIP-based approach DeepCIP.

Word2Vec Embeddings. Neural word embeddings, introduced in [19, 135], are learned vector representations for each word from a text corpus. These neural word embeddings are useful for predicting the surrounding words in a sentence. A common approach is to use a multi-layer Skip-gram model with negative sampling. The objective function minimizes the distance of each word with its surrounding words within a sentence while maximizing the distances to randomly chosen set of words (*negative samples*) that are not expected to be close to the target. This is an objective quite similar to ours as it enables to compute proximity between items in the same CIP. This approach computes similarity between two words as the dot product of their word embeddings.

DeepCIP Algorithm. We now describe how the Word2Vec concept is adapted to CIPs, for they allow scalable and fresh item incorporation in the model. We feed a skip-gram model with item-pairs in CIPs where each CIP is as usual an ordered set of items (similar to the instantiation in CIP-I). More precisely, CIPs are δ-distant as instantiated for CIP-I. DeepCIP trains the neural network with pairs of items at a distance less than a given *window size* within a CIP. This window size corresponds to the notion of hammock distance (as defined for CIP-U) where the distance hyper-parameter δ_H is defined by the *window size*. More formally, given a sequence of T training items' vectors i_1, i_2, i_3, ..., i_T, and a maximum hammock distance of k, the objective of the DeepCIP model is to maximize the average log probability.

$$\frac{1}{T}\sum_{t=k}^{T-k} log\, P(i_t|i_{t-k},....,i_{t-1},i_{t+1},....,i_{t+k}) \tag{4.15}$$

The Skip-gram model is employed to solve the optimization objective 4.15 where the weights of the model are learned using backpropagation and stochastic gradient descent (SGD). SGD is inherently synchronous as there is a dependence between the update from one iteration and the computation in the next iteration. Each iteration must potentially wait for the update from the previous iteration to complete. This approach does not allow the distribution of computations on parallel resources which leads to a scalability issue. To circumvent this scalability issue, we implement DEEPCIP using asynchronous stochastic gradient descent (DOWNPOUR-SGD [48]). DOWNPOUR-SGD enables distributed training for the skip-gram model on multiple machines by leveraging asynchronous updates from them. We use a publicly-available deep learning framework [50] which implements DOWNPOUR-SGD in a distributed setting. More precisely, DEEPCIP trains the model using DOWNPOUR-SGD on the recent CIPS thereby updating the model incrementally.

DEEPCIP uses a *most_similar* functionality to select items to recommend to a user, using as input recently consumed items (current CIP). We compute a CIP vector using the items in the given CIP and then use this vector to find most similar other items. More precisely, the *most_similar* method uses the cosine similarity between a simple mean of the projection weight vectors of the recently consumed items (i.e., items in a user's most recent CIP) and the vectors for each item in the model.

Incremental updates. Online machine learning is performed to update a model when data becomes available. The DEEPCIP model training is performed in an online manner [63] where the model is updated using the recent CIPS. Online machine learning is crucial in recommendation as it is necessary for the algorithm to dynamically adapt to new temporal patterns [37] in the data. Hence, the complexity of the model update is dependent on the number of new CIPS received along with the hyper-parameters for the learning algorithm (primarily, skip-gram model parameters, dimensionality of item vectors, number of training iterations, hammock distance).

4.2.4 Implementation

We provide here some implementation details of our CIP-based algorithms, i.e.,CIP-U, CIP-I and DEEPCIP.

A. Spark Data Structures

We consider Apache Spark [172] as our framework for recommendation computations. Spark is a cluster computing framework for large-scale data processing. It is built on top of the Hadoop Distributed File System (HDFS) and provides several core abstractions, namely Resilient Distributed Datasets (RDDs), parallel operations and shared variables.

An RDD is a fault-tolerant abstraction that enables users to explicitly persist intermediate results in memory and control their partitioning to optimize data placement. It is a read-only

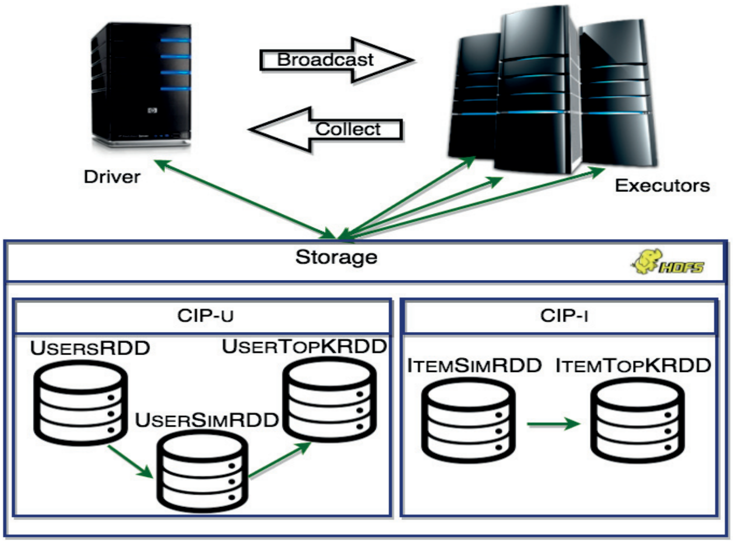


Figure 4.15 – *Topology and data structures for* CIP-U *and* CIP-I *(arrows denote the* RDD *dependencies).*

collection of objects partitioned across a set of machines and can be rebuilt if a partition is lost. In a Spark program, data is first read into an RDD object. This RDD object can be altered into other RDD objects by using *transformation operations* like `map`, `filter`, and `collect`. Spark also enables the use of shared variables, such as *broadcast* and *accumulator*, for accessing or updating shared data across worker nodes.

B. Tailored Data Structures for CIPs

We now mention briefly the RDDs leveraged in the memory-based approaches (CIP-U and CIP-I) as shown in Figure 4.15 (the arrows, between RDDs, in the figure denotes the sequential dependency between the RDDs through transformation operations) as well as those in the model-based approach (DEEPCIP) as shown in Figure 4.16.

RDDs for CIP-U. For CIP-U, we store the collected information into three primary RDDs as follows. USERSRDD stores the information about the user profiles. USERSIMRDD stores the hammock pairs between all pairs of users. The pairwise user similarities are computed using a transformation operation over this RDD. USERTOPKRDD stores the K most similar users.

During each update step in CIP-U, after Q consumption events, the new events are stored into a DELTAPROFILES RDD which is broadcast to all the executors using the *broadcast* abstraction of Spark. Then, the hammock pairs between users are updated (in USERSIMRDD) and consequently transformed to pairwise user similarities using Equation 4.13. Finally, CIP-U updates the the top-K neighbors (USERTOPKRDD) based on the updated similarities.

RDDs for CIP-I. For CIP-I, we store the collected information into two primary RDDs as follows. ITEMSIMRDD stores *score* values between items. The pairwise item similarities are computed using a transformation operation over this RDD. ITEMTOPKRDD stores the K most

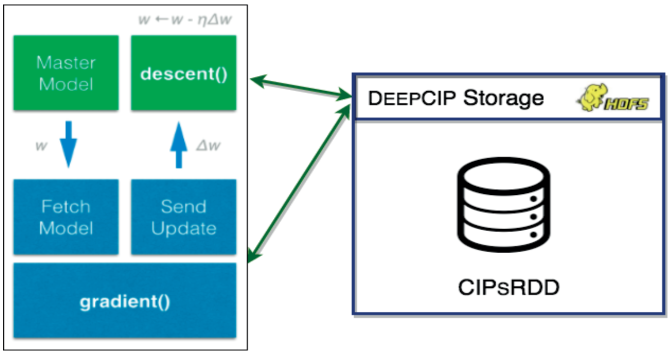


Figure 4.16 – *Topology and data structures for* DeepCIP.

similar items for each item based on the updated similarities.

During each update step in CIP-i, the item scores are updated incorporating the received CIP using Algorithm 4 in the ItemSimRDD, and consequently the pairwise item similarities are also revised using Equation 4.14. CIP-i computes the top-K similar items and updates the ItemTopKRDD at regular intervals.

RDDs for DeepCIP. We implement the DeepCIP using the DeepDist deep learning framework [50] which accelerates model training by providing asynchronous stochastic gradient descent (Downpour-SGD) for data stored on Spark.

DeepCIP implements a standard master-workers parameter server model [48]. On the master node, the CIPsRDD stores the recent CIPs aggregated from the user transaction logs preserving the consumption order. DeepCIP trains on this RDD using the Downpour-SGD. The skip-gram model is stored on the master node and the worker nodes fetch the model before processing each partition, and send the gradient updates to the master node. The master node performs the stochastic gradient descent (Equation 2.8 in §2.5) asynchronously using the updates sent by the worker nodes. Finally, DeepCIP predicts the most similar items to a given user, based on her most recent CIP.

4.2.5 Evaluation

In this section, we report on the evaluation of the CIP-based algorithms, using real-world datasets.

Platform. For our experiments, we use two deployment modes of the Spark large-scale processing framework [172].

Standalone deployment. We launch a Spark Standalone cluster on a highperf server (Dell Poweredge R930) with 4 Processors Intel(R) Xeon(R) E7-4830 v3 (12 cores, 30MB cache, hyper-threading enabled) and 512 GB of RAM. We use this cluster to evaluate the effect of the number of partitions for the RDD on scalability. For the standalone deployment, we use 19 executors each with 5 cores since we have a total of 96 cores in this cluster.[17]

YARN deployment. We use the Grid5000 testbed to launch a Spark cluster consisting of 20 machines on Hadoop YARN. Each machine is an Intel Xeon CPU E5520@ 2.26GHz. For the YARN deployment, we set the number of executors equal to the number of machines in the cluster.

Datasets. We use real-world traces from a movie recommendation website: MovieLens (ML-100K, ML-1M) [138] as well as a product review website: Ciao [40]. Those traces contain users' ratings for movies they enjoyed. We compare the performance of our implicit CIP based models to the one of a widespread explicit (rating-based) collaborative filtering. In these datasets, each user rated at least 20 movies. The ratings vary from 1 to 5 with an increment of 1 between the possible ratings. Note that the ratings are only used for the explicit (rating-based) recommender. Table 4.3 provides further details about these datasets along with their *densities*. The density of a dataset denotes the fraction of actual user-item (implicit or explicit) interactions present in the dataset compared to all the possible interactions.

Datasets	#Users, #Items	#Training, #Validation, #Test	Density
ML-100K	943, 1682	75000, 5000, 20000	6.31%
ML-1M	6040, 3952	970209, 10000, 20000	4.19%
Ciao	489, 12679	19396, 1000, 2000	0.36%

Table 4.3 – *Details of the datasets used in our experiments.*

Metrics. We evaluate the recommendation quality in terms of the *Precision* (§2.7) which is a classification accuracy metric used conventionally to evaluate top-N recommenders [43]. Precision denotes the fraction of recommended items which were indeed relevant to the target user.

Hyper-parameters. We tune the core hyper-parameters for CIP-U, CIP-I and DEEPCIP. For CIP-U, we have the *hammock threshold* (δ_H) whereas for the CIP-I, we have the distance (δ) to separate δ-distant CIPS in a user's profile. For DEEPCIP, we have the distance (δ), similar to CIP-I, as well as the window size (W) which denotes the maximum hop allowed for learning the item vectors within a CIP. These hyper-parameters essentially determine the optimal size of the consumption interval for achieving the best recommendation quality.

Evaluation scheme. The dataset is sorted based on the unix timestamps associated with the rating events. Then, the sorted dataset is replayed to simulate the actual temporal behavior of users. We measure the recommendation quality as follows: we divide the dataset into a *training set*, a *validation set* and a *test set*. The training set is used to train our CIP based models whereas the validation set is used to tune the hyper-parameters of the models. For each event in the test set (or rating when applied to explicit recommenders), a set of top

---

[17] We use this deployment for running long duration experiments, due to reservation limitations on the Grid5000 cluster [73].

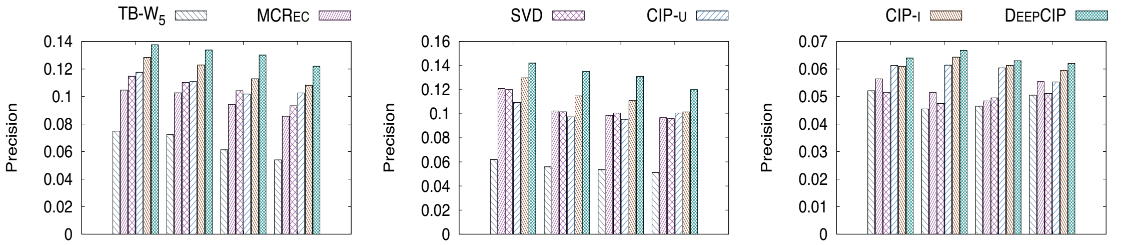
recommendations is selected as the *recommendation set* with size denoted as N. Note that we recommend the most popular items for new users (cold-start). Table 4.3 shows the partition between training, validation and test sets along with the details of the datasets.

Competitors. We compare the recommendation quality of our three algorithms with also three competitors: a *matrix factorization* based technique (using explicit ratings) [111], a popular time-based recommender (without using any explicit ratings) [115], and the state-of-the art approach mixing both implicit and explicit information [82].

Matrix factorization. Matrix factorization techniques map both users and items to a joint latent factor space of dimensionality f, such that ratings are modeled as inner products in that space. We use a publicly available library (Python-recsys [147]) for empirical evaluations. Python-recsys is a widely used recommender framework for SVD-based approaches [191, 169, 178].

Implicit time-based recommender. We compare with a popular time-based recommender designed to provide recommendations without the need for explicit feedback [115]. They construct pseudo ratings from the collected implicit feedback based on temporal information - *user purchase-time* and *item launch-time* - in order to improve recommendation accuracy. They use two rating functions: W_3 (coarse function with three launch-time groups and three purchase-time groups) and W_5 (fine-grained function with five launch-time groups and five purchase-time groups) where the later performs slightly better. Hence, we choose W_5 rating function for our empirical comparison and we denote this system as $TB-W_5$ in our evaluation.

Markov chain-based recommender. We compare with a recent recommender which combines matrix factorization and markov chains [153, 82] to model personalized sequential behavior. We use a publicly available library [161] for our empirical evaluation. We denote this system as MCREC in our evaluation.



(a) *ML-100K* **(b)** *ML-1M* **(c)** *Ciao*

Figure 4.17 – *Recommendation quality of CIP-based algorithms versus competitors.*

Quality comparison with competitors. Once we obtain the optimal setting of the hyper-parameters for our CIP based models, we compare them with the competitors namely: the matrix factorization based technique (SVD), the markov-chain based technique (MCREC) and the time-based approach (TB-W_5). We compare the recommendation quality in terms of the precision ($N = 10$) on MovieLens (ML-100K, ML-1M) and Ciao datasets, in Figure 4.17. We draw the following observations.

- Regarding our three algorithms, DEEPCIP always outperforms CIP-I, which in turn is always outperforming CIP-U (except on the Top-5 result on the Ciao dataset which is due to the relatively limited number of recommendations).
- The CIP-based algorithms outperform TB-W_5 on all the three datasets. For example, consider top-10 recommendations in the ML-1M dataset, CIP-U provides around 1.82× improvement in the precision, CIP-I provides around 2.1× improvement, and DEEPCIP provides around 2.4× improvement.
- The CIP-U algorithm performs on par with MCREC as well as matrix factorization based techniques. CIP-I overcomes MCREC on all three scenarios, sometimes only by a short margin (ML-1M). However, the DEEPCIP model outperforms all other models significantly. For example, consider the top-10 recommendations in the ML-1M dataset, DEEPCIP provides 2.4× improvement over TB-W_5, 1.29× improvement over MCREC, and 1.31× improvement over the matrix factorization based one. The reason behind this improvement is that DEEPCIP considers, for any given item, the *packs* of items at a distance dependent on the defined window size, whereas MCREC only considers pairs of items in the sequence of chain states (and thus has a more constrained learning process).

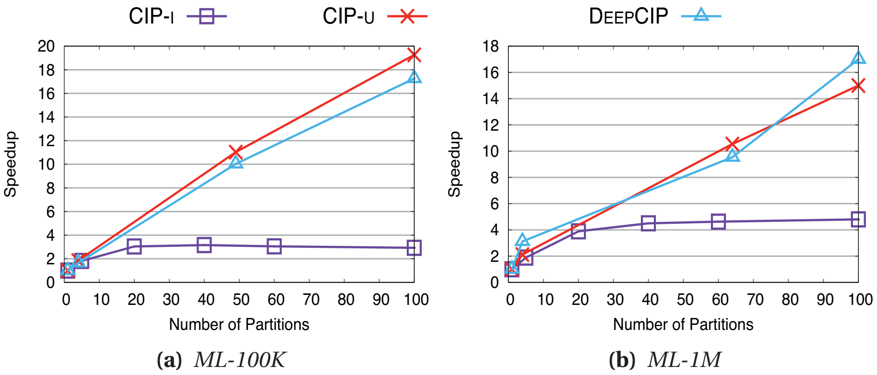
Note that the precision we obtain for SVD on MovieLens (11% to 12%) is consistent with other standard quality evaluation benchmarks for state-of-the-art recommenders [43].

These results show the existence of the latent information contained in closely consumed items, accurately captured by the CIP structure. Note that this is intuitively consistent for DEEPCIP to perform well in this setting: the original WORD2VEC concept captures relation among words w.r.t. their proximity in a given context. With DEEPCIP, we seek to capture item proximity w.r.t. their consumption time.

Scalability. We now evaluate the scalability of CIP-based algorithms by varying the number of RDD partitions employed by Spark as well as the size of the Spark cluster.

Effect of partitions. Spark's RDD deals with fragmented data which enables Spark to efficiently execute computations in parallel. The level of fragmentation is a function of the number of partitions of an RDD which is crucial for the scalability performance of an application. A small number of partitions reduces the concurrency and consequently leads to underutilization of the cluster. Furthermore, since with fewer partitions there is more data in each partition, this increases the memory pressure on the application. On the flip side, with too many partitions, the performance might degrade due to data shuffling as it takes a hit from the network overheads and disk I/Os. Hence, tuning the number of partitions is important in determining the attainable scalability of an algorithm. We thus conduce the effect of the number of partitions on scalability. We run these experiments in the Standalone mode of Spark.

Figures 4.18a and 4.18b demonstrate that scalability depends on the number of partitions which is ideally equal to the number of cores in the cluster. We observe a near-linear speedup while increasing the number of partitions for both CIP-U as well as DEEPCIP. However, the speedup is comparatively less for CIP-I due to the highly reduced time complexity of CIP-I leading to significantly less computations.



(a) *ML-100K* **(b)** *ML-1M*

Figure 4.18 – *Partition effects.*

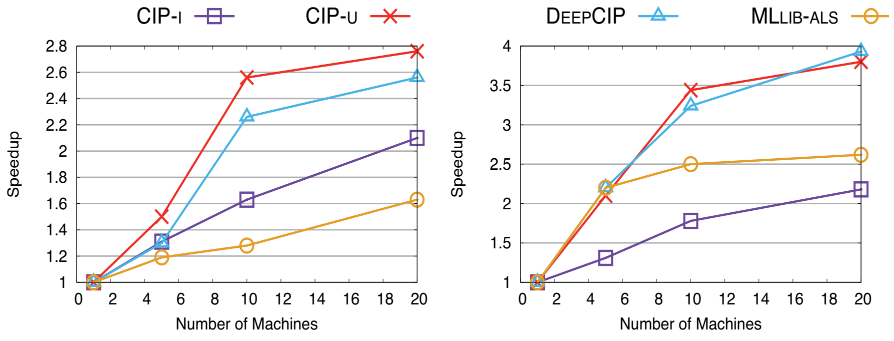
Effect of Cluster size. We now evaluate the scalability of our algorithms while increasing the cluster size from one machine to a maximum of 20 machines. Furthermore, we also compare the speedup achieved by a matrix factorization technique (ALS) implemented in the publicly available MLLIB library for Spark. Number of partitions is set to 50.

Figure 4.19 depicts a sublinear increase in speedup while increasing the number of machines on both the datasets. The sublinearity in the speedup is due to communication overheads in Spark with increasing number of machines. The speedup on ML-1M is higher due to more computations being required for larger datasets and higher utilization of the cluster. We observe that the speedup for CIP-I is similar for both datasets as its time complexity depends on the CIP size (Algorithm 4). DEEPCIP scales well due to the distributed asynchronous stochastic gradient descent (DOWNPOUR-SGD) for training the skip-gram model where more gradient computations could be executed asynchronously in parallel with increasing number of nodes. CIP-U and DEEPCIP scale better than ALS for both setups.

4.2.6 Related Work

We now discuss previous work about using explicit and implicit feedback in recommenders.

Explicit feedback. Tapestry [68], one of the earliest implementations of collaborative filtering, relies on the explicit opinions of people from a close-knit community such as an office working group. Since then, a lot of work has been devoted to improve the recommendation quality. All however require explicit feedback like numerical ratings, binary like/dislike or just positive likes. Recently, Sen et al. demonstrated that different rating scales elicit different levels of cognitive load on the end users [173]. Whitenton pointed out the relation between



(a) *ML-100K* **(b)** *ML-1M*

Figure 4.19 – *Cluster size effects.*

cognitive load and consumer usability and highlighted the very fact that to achieve maximum usability, the cognitive load should be minimized [184]. In this work, we focus on utilizing the information available in transaction logs, for it is available to arguably all services proposing a catalog of items.

Implicit feedback. Our CIP-based algorithms belong to the category of recommenders using implicit feedback from users [141]. HOSLIM [39] proposes to compute higher order relations between items in consumed itemsets; those relations are the ones that maximize the recommendation quality, but without notions of temporality in item consumption. The proposed algorithm is time-agnostic, and does not scale for orders superior to pairs of items. Moreover, it is not designed to efficiently incorporate freshly consumed items and suffers from computational intractability. Barkan et al. present ITEM2VEC [16], that also uses skip-gram with negative sampling to retrieve items' relations w.r.t their context in time. Besides the fact that their implementation does not scale on multiple machines due to the use of synchronous stochastic gradient descent, they evaluated only on private datasets. This makes precise evaluations w.r.t. state-of-the-art algorithms subjective. Implicit feedback has also been used for multiple other applications: this is traditionally the case in search engines, where clicks are tracked [42]. SPrank [142] leverages semantic descriptions of items, gathered in a knowledge base available on the web. Koren et al. [90] showed that implicit information, like channel switching on TV, is valuable enough to propose recommendations. Huang et al. leverage unordered co-occurrence of contextual queries in session-based query logs in a non-incremental manner for relevant term suggestion in search engines [91]. Recommenders can also use the implicit social information of their users to improve final results [128].

Interestingly enough, in the context of music recommendation, Jawaheer et al. [95] pointed out that implicit and explicit recommenders are complementary, and experimentally perform similarly. Recently, Soldo et al. leveraged users' malicious (implicit) activity logs to recommend which IP addresses to block [171]. Hence, implicit feedback based approaches could be employed over a wide range of applications.

Time-based recommendation. Within implicit based recommenders, the notion of "time" has been exploited in various ways since it is a crucial implicit information collected by all services. Some companies implement implicit recommenders, as *e.g.*, Amazon [10]; yet, we are not aware of the use of any technique even remotely close to our notion of item packs. The use of spatio-temporal proximity between users in a given place was introduced in [47]. However, such a technique requires auxiliary location-based information for detecting such user proximity, which furthermore might be a privacy concern for users (*location privacy* [17]). Baltrunas et al. presented a technique [13] very similar to CIP where a user profile is partitioned into micro-profiles (similar to CIPS in our approach). However, explicit feedback is required for each of these micro-profiles, to improve the quality of recommendations. Time window (or decay) filtering is another technique, applied to attenuate recommendation scores for items having a small likelihood to be purchased at the moment when a user might view them [70]. While such an approach uses the notion of time in transaction logs to improve recommendations, it still builds on explicit ratings for computing the basic recommendation scores. Campos et al. [32] proposed to bias recommendation according to freshness of ratings in the dataset. However, their approach still uses explicit ratings to improve recommendation quality using their time-biased strategy. Finally, Lee et al. [115] introduced a completely implicit feedback based approach that gives more weight to new items if users are sensitive to the item's launch times. We compare our algorithms to this approach in §4.2.5 and demonstrate that our CIP-based algorithms perform better in practice.

Sequence-based recommendation Recently, there have been some approaches using Markov chains to model consumption sequences [153]. However, such approaches suffer from sparsity issues and the long-tailed distribution of many datasets. We compare with a Markov-chain based approach (MCREC) and show that CIP-based approaches, updated incrementally in a distributed manner, perform on par with MCREC.

4.2.7 Conclusion

Since very recently, research efforts are dedicated to circumvent the absence of explicit feedback on online platforms, using individual techniques that leverage the sequential consumption of items. In an effort for a detailed and scalable proposal for generalizing such a direction, we presented two memory-based and one model-based recommendation algorithms exploiting the implicit notion of *item packs consumed by users*, while showing that our framework can also incorporate the previous state-of-the-art approach on the topic. Our novel algorithms provide a better recommendation quality than the widespread SVD-based approach [111], as well as implicit ones leveraging consumption time [115] or consumption sequences [82, 153]. This confirms the fact that item packs allow to efficiently identify similar users or items. Importantly, for practical deployments, this key latent feature can be captured with the incremental algorithms that we presented, thus allowing to build fast services using freshly consumed items. Deeper analysis might be conduced in a sociological direction, in order to validate further the relevance and robustness of this latent feature, across different datasets and services.

PART III

Privacy

Personalization and privacy are two sides of the same coin in the sense that there is a significant underlying trade-off between these two aspects. Personalization improves with an increase in the amount of data. However, data leaks information about users and hence leads to severe privacy concerns. In this part of the thesis, we will see how we can protect privacy of the users while providing personalized recommendations to them. We consider two levels of privacy.

- In §5.1, we first focus on protecting the privacy of any user from other curious users, which we denote as *user-level privacy*.
- We next provide a brief overview, in §5.2, of our approach to protect the privacy of users from the service provider itself, which we denote as *system-level privacy*.

5 Privacy

5.1 User-level Privacy

5.1.1 Overview

CF recommenders induce an inherent trade-off between privacy and personalization [119]. In this work, we address this trade-off by exploring a promising approach where the information used for computing recommendations is *concealed*. We present D2P, a novel protocol that uses a probabilistic substitution technique to create the AlterEgo profile of an original user profile. D2P ensures a strong form of *differential privacy* [55, 57], which we call **D***istance-based* **D***ifferential* **P***rivacy*. Differential privacy [55, 57] is a celebrated property, originally introduced in the context of databases. Intuitively, it ensures that the removal of a record from a database does not change the result of a query to that database - modulo some arbitrarily small value (ϵ). In this sense, the presence in the database of every single record - possibly revealing some information about some user - is anonymous as no query can reveal the very existence of that record to any other user (modulo ϵ). Applying this notion in the context of recommenders would mean that - modulo ϵ - no user v would be able to guess - based on the recommendations she gets - whether some other user u has some item i in her profile, e.g., whether u has seen some movie i. Such a guarantee, however, might be considered too weak as nothing would prevent v from guessing that u has in her profile some item that is very similar to i, e.g., that u has seen some movie similar to i.

We strengthen the notion of differential privacy in the context of CF recommenders to guarantee that any user v is not only prevented from guessing whether the profile of u contains some item i, but also whether the profile of u contains any item i' within some *distance* λ from i (say any movie of the same category of i): hence the name **D***istance-based* **D***ifferential* **P***rivacy* (D2P). Our D2P protocol ensures this property.

The basic idea underlying D2P is the following. We build, for each user profile, an *AlterEgo* profile corresponding to it. The latter profile is based on the former one where we probabilistically replace some of the items with either related or random ones. This poses of course a challenging technical problem. If the *AlterEgo* profile is too far from the original one, the recommendation quality is impacted: we lose the benefits of collaborative filtering. If the profile is too close to the original one, privacy remains weak. We demonstrate in this work that the quality of the D2P recommendation is still good for values of λ that can hide items within a reasonable distance from the original profile - what might be considered a reasonable distance depends on the dataset as we explain later in this work.

To illustrate the basic idea, consider traces from MovieLens and the scenario of Figure 5.1, with a total of 5 movies. Consider a user who likes Shawshank Redemption (SR). We compute the distance between the other 4 movies from SR based on their similarity (as shown later in Equation 5.2). D2P selects movies (for replacement) with distance less than the upper bound ($\lambda = 0$, 1 or 2) with high probability (p) and any random movie from the dataset, including those close to the item to be replaced, with a low probability ($1 - p$). If λ is set to 0, then D2P satisfies the classical differential privacy (with ϵ given in Equation 5.3 in §5.1.3). Our results in §5.1.4 show that even if we consider λ as 6.5, we still have a good recommendation quality.

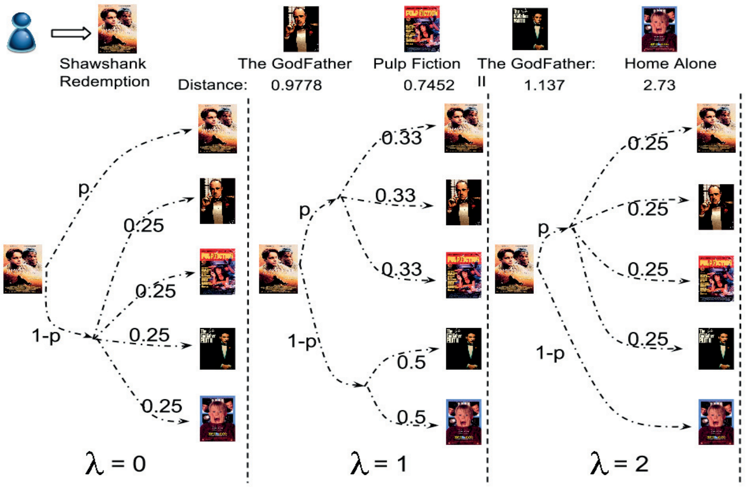


Figure 5.1 – D2P *Illustration.*

D2P provide formal privacy guarantees in terms of parameters ϵ and λ. We also provide a through empirical evaluation of the privacy-quality trade-off on real-world datasets, namely MovieLens and Jester. Our results show that D2P provides proved privacy guarantees while preserving the quality of the recommendation. We demonstrate, for instance, that D2P achieves 1.5 times the coverage [65] provided by a standard recommender for MovieLens dataset. Additionally, we show that the privatization overhead in D2P is very small compared to [132], which makes it appealing for real-time workloads.

Interestingly, D2P is a generic protocol. As we show through our performance results, it applies well in the context of a user-based and an item-based recommender. D2P can also be customized for recommendation infrastructures where a KNN computation is deployed either on the cloud [148] or on user machines [27].

5.1.2 D2P: Privacy for Recommenders

Preserving privacy in CF recommenders is challenging. It was shown using the Netflix Prize dataset that even anonymizing individual data before releasing it publicly is not enough to preserve privacy [139]. Even cryptographic approaches do not preclude the possibility of the output leaking information about the personal input of individuals [181]. The need for stronger and robust privacy guarantees motivated the emergence of the notion of *Differential Privacy* [55, 57, 64]. First introduced in the context of databases, differential privacy provides quantifiable privacy guarantees. We introduce a stronger form of this notion in the context of recommenders by accounting for the concept of distance between items.

A. Differential Privacy

Differential Privacy (DP) implies that the output of a given function becomes significantly more or less likely - based on some parameter ϵ - if the inputs differ in one record. The basic intuition is that an observer can extract limited information from the output in the absence or presence of a specific record in the database.

Definition 8 (DIFFERENTIAL PRIVACY)**.** *A randomized function* $\mathcal{R}$ *provides* ϵ*-differential privacy if for all datasets* $\mathcal{D}_1$ *and* $\mathcal{D}_2$*, differing on at most one element, and all* $\mathcal{S} \subseteq Range(\mathcal{R})$*, the following inequality always holds:*

$$\frac{Pr[\mathcal{R}(\mathcal{D}_1) \in \mathcal{S}]}{Pr[\mathcal{R}(\mathcal{D}_2) \in \mathcal{S}]} \leq e^{\epsilon}$$

Here, e^{ϵ} denotes $exp(\epsilon)$.

B. Distance-based Differential Privacy

With differential privacy applied in its classical form recalled above to a recommender, an adversary (a curious user) cannot know if one item has been rated by a user. However, the adversary can know about items similar to the rated ones. Hence, the adversary can infer fairly accurate information about user preferences without knowing the exact items rated by that user. In this sense, classical differential privacy is not enough in the context of a recommender.

Our notion of *Distance-based Differential Privacy* is stronger: it extends DP to recommenders. We ensure differential privacy for all the items, rated by that user, and ones that are within a distance of λ. The distance parameter (λ) determines the *closely related items to form the AlterEgo profiles*, thereby *concealing* the actual user profiles and preferences. The distance parameter also aids in tuning the recommendation quality using the *AlterEgo* profiles as shown later in Figure 5.10.

It is important to notice that our notion of Distance-based differential privacy is independent from the underlying recommendation algorithm used. To define this new notion more precisely, we first define the notions of *Distance-based Group* and *Adjacent Profile Sets.*

Definition 9 (ELEMENT-WISE GROUP)**.** *We denote by* $\mathbb{E}$ *the set of all elements. For every element* $x \in \mathbb{E}$*, distance function* $\Lambda : \mathbb{E} \times \mathbb{E} \rightarrow \mathbb{R}^{+} \cup \{0\}$*, and fixed distance threshold* λ*, then* $\mathcal{GRP}_{\lambda}(x)$ *is defined as the collection of all elements* $x_k \in \mathbb{E}$ *such that* $\Lambda(x, x_k) \leq \lambda$*. More specifically:*

$$\mathcal{GRP}_{\lambda}(x) = \{x_k \in \mathbb{E} | \Lambda(x, x_k) \leq \lambda\}$$

We extend this notion of groups to a set of elements where each element in the set has a *Group* defined by Definition 9.

Definition 10 (SET-WISE GROUP)**.** *For a set of elements* $\mathcal{S}$*,* $\mathcal{GRP}_{\lambda}(\mathcal{S})$ *is the union of all the groups:* $\mathcal{GRP}_{\lambda}(s)$ *for each element* s$\in \mathcal{S}$*. More specifically:*

$$\mathcal{GRP}_{\lambda}(\mathcal{S}) = \bigcup_{s \in \mathcal{S}} \mathcal{GRP}_{\lambda}(s)$$

We now introduce the notion of *Neighboring Groups* (used in §5.1.3).

Definition 11 (NEIGHBORING GROUP)**.** *We define the* KNN *groups* (KNN($\mathcal{GRP}_{\lambda}(x)$)) *of* $\mathcal{GRP}_{\lambda}(x)$ *for an element* x *as the* $Top - \mathcal{K}$ *groups sorted in decreasing order by the count of shared elements with* $\mathcal{GRP}_{\lambda}(x)$*.*

Definition 12 (ADJACENT PROFILE SET)**.** *An event in the context of* D2P *is an interaction between the system and the user when the user provides a rating for some item in the system. Two profile sets* $\mathcal{D}_1$ *and* $\mathcal{D}_2$ *as* adjacent profile sets *when* $\mathcal{D}_1$ *and* $\mathcal{D}_2$ *differ in only one event, which implies one user-item rating pattern is different in these two profile sets.*

For any arbitrary recommendation mechanism $\mathcal{R}$, which takes a profile set and a specific user as input, the output is the set of items that the algorithm recommends to that specific user.

Definition 13 (DISTANCE-BASED DIFFERENTIAL PRIVACY)**.** *For any two adjacent profile sets* $\mathcal{D}_1$ *and* $\mathcal{D}_2$*, where* u *denotes any arbitrary user and* $\mathcal{S}$ *denotes any possible subset of elements, then any mechanism* $\mathcal{R}$ *is* (ϵ, λ)*-private if the following inequality holds:*

$$\frac{Pr[\mathcal{R}(\mathcal{D}_1, u) \in \mathcal{GRP}_{\lambda}(\mathcal{S})]}{Pr[\mathcal{R}(\mathcal{D}_2, u) \in \mathcal{GRP}_{\lambda}(\mathcal{S})]} \leq e^{\epsilon} \tag{5.1}$$

The result of the recommendations for two profile sets that are close to each other are of the same order probabilistically with a coefficient of e^{ϵ}. Later in §5.1.3, we present the mathematical relationship between ϵ and λ.[1] D2P conceals the profiles by anonymizing elements within distance λ from the elements of the original profile. We get the classic notion of differential privacy with λ as 0. If we increase λ then the privacy increases but the quality decreases slightly as shown later in Figure 5.10(a). In a user-level privacy scheme, more than one event can differ for a profile in two adjacent profile sets, whereas in an event-level privacy approach a single event differs for a profile in two adjacent profile sets.

5.1.3 D2P-based Recommender

Our D2P-based recommender implements a variant of the general CF recommendation scheme, based on KNN (K Nearest Neighbors [175]), incorporating the D2P protocol. The working principle of such a scheme is twofold (Algorithm 1). Firstly, the k most similar *neighbors* of any active user are identified in the KNN selection phase. Secondly, the recommendation algorithm is run to suggest items to the users leveraging the profiles obtained through the KNN selection.

We consider a recommender scheme that stores *user profiles* and *item profiles*. The profile of a user u, denoted by P_u, consists of all the items rated (alternatively shared or liked) by u along with the ratings. In our implementation, we convert the numerical ratings into binary ratings, a *like* (1) or a *dislike* (0). [2] An *item profile* (P_i) consists of users who rated item i along with the ratings.

D2P relies on the *distance* between items to create *AlterEgo* profiles, as we discuss below. The recommender in D2P operates in four phases as shown in Figure 5.2.

A. Grouping Phase

In this phase, groups are formed for each item: group G_i for item i contains all the items with distance less than a predefined upper-bound λ. In our scheme, we define the distance $\Lambda_{i,j}$ between items i and j as:

$$\Lambda_{i,j} = \frac{1}{\Psi(i,j)} - 1 \tag{5.2}$$

Here, $\Psi(i,j)$ denotes the cosine similarity between items i and j. The neighboring group G_j of a group G_i is defined as a group with which group G_i shares at least one item. Groups can also be formed based on item features (e.g. genres, date-of-release in case of movies) where similarity is measured between the feature vectors of the items. The groups need to be updated periodically to account for newly added items and ratings. In D2P, the grouping of the items in the *Grouping Phase* is performed by the *FormGroups* function shown in Algorithm 5. An item can be included in more than one groups, e.g., an *action-comedy* movie X can be present

---

[1] For more details regarding the correctness proofs of our privacy guarantee (Definition 13), we refer to our paper [76] for interested readers.

[2] Binary ratings are considered for the sake of simplicity: this scheme can be generalized to numerical ratings.

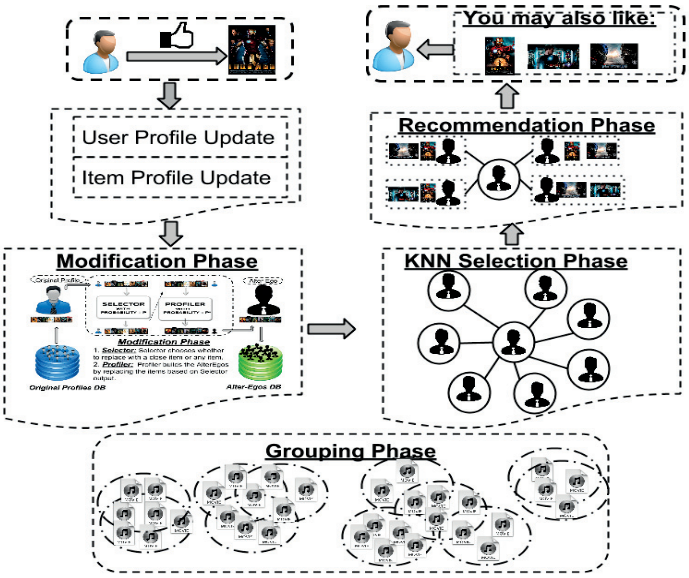


Figure 5.2 – D2P*-based Recommender.*

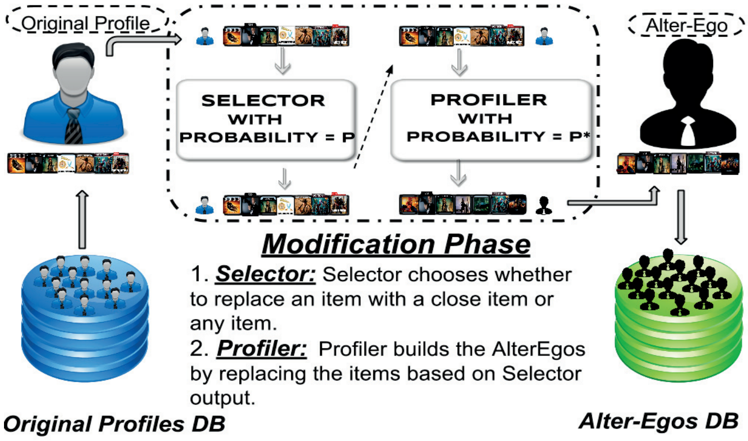
in the group of an *action* movie as well as in the group of a *comedy* movie.

Algorithm 5 Grouping : *FormGroups(ItemSet): Grouping Phase where ItemSet is the set of all items in the database*

```
Parameter: $\lambda$                                  ▷ Distance threshold
var $ItemSet$;                                        ▷ Denotes set of all items in the network
var $\lambda$;                                        ▷ Distance Metric
for $i$ : item in $ItemSet$ do
    $Group_i.add(i)$;
    for $j$ : item in $(ItemSet \setminus i)$ do
        $S = \Psi(i, j)$;                             ▷ Compute Similarity
        if $S > 0$ then
            $\Lambda_{i,j} = (1/S) - 1$;
            if $\Lambda_{i,j} \leq \lambda$ then
                $Group_i.add(j)$;
            end if
        end if
    end for
end for
return: $Group$;                                      ▷ The groups for the items
```

B. Modification Phase

Privacy breaches occur in any standard user-based CF recommender due to leakage of the information of neighboring profiles to any active curious user through recommendations provided to her. D2P relies on the above-mentioned groups of items, generated in the previous phase, to create *AlterEgo* profiles, and thus avoids to reveal the exact ones. D2P protects the privacy of users in the modification phase employing two components (conveyed by Figure 5.3): *Selector*, which selects the items to replace, and the *Profiler*, which determines by which items those entries should be replaced. These two components conceal the neighbors' information from the active user, preventing this user to correlate the recommendations to the neighbors' profiles. The *selector* and *profiler* are responsible for generating the *AlterEgo*[3] profiles in such a way that the quality is not impacted too much while privacy is preserved. We now provide details on these two core components.



1. ***Selector:*** Selector chooses whether to replace an item with a close item or any item.
2. ***Profiler:*** Profiler builds the AlterEgos by replacing the items based on Selector output.

Figure 5.3 – D2P *Modification Phase.*

D2P Selector. The *selector* is responsible for selecting the items to replace by the *profiler* to form the *AlterEgo* profiles. We select an item with a probability p to replace with any possible item at random and with a probability $1-p$ to replace with some random item from the respective group (and neighboring groups) for that respective item. The *getSelectProb* function mentioned in Algorithms 6 and 7, returns a random real number between 0 and 1. Finally, the *selector* outputs a set of actual items (*GItems*) to be replaced by *GroupItems* and another set of actual items (*RItems*) to be replaced by any item from the set of all possible items at random.

Algorithm 6 Selector Algorithm: *Selector*(P_u) *where* P_u *is the profile of user u*

```
Parameter: $p$                                   ▷ Selector Probability
var $GItems[u] = NULL$                           ▷ Replace with group item
var $RItems[u] = NULL$                           ▷ Replace with any item
for $i$ : item in $P_u.getItems()$ do
    if $getSelectProb() > p$ then
        $GItems[u] = GItems[u] \cup i$;
    end if
    if $getSelectProb() \leq p$ then
        $RItems[u] = RItems[u] \cup i$;
    end if
end for
return: {$GItems[u]$, $RItems[u]$};
```

D2P Profiler. The *profiler* builds the *AlterEgo* profiles which are used in the KNN selection phase. The *profiler* replaces items in *GItems* with items from their respective group (and

---

[3] The *AlterEgo* profile of a user u denotes the imitation profile of u which hides the user preferences by substituting items in the user profile by utilizing D2P.

neighboring groups) with a probability $1-p^*$ and retains the original item with a probability p^*. We also substitute items in *RItems* with items from the set of all possible items with a probability $1-p^*$ and preserves the actual ones with a probability p^*. The $SRSI$ (Select Random Set Item) function in Algorithm 7 selects randomly an item from the respective groups' items. It selects either from *GroupItems* (based on a distance metric between items) for all the items in the set *GItems* or from the *ItemSet* for all the items in *RItems*.

Algorithm 7 Profiler Algorithm: *Profiler*(P_u) *where* P_u *is the profile of user u*

```
Parameter: $p^*$                                          ▷ Profiler Probability
var {GItems[$u$], RItems[$u$]} = Selector($P_u$);
var Items[$u$] = $GPI(P_u)$                                ▷ Get items from $P_u$
var ItemSet;                                               ▷ Set of all items in the network
for $i$ : item in $P_u$.getItems() do
    GroupID = $Group_i$;
    NBGroupIDs = $Group_i$.getNeighbors();
    Groups = GroupID ∪ NBGroupIDs;
    GroupItems = $\bigcup_{G \in Groups}$ Group.get($G$);
    if (getSelectProb() > $p^*$ & $i \in$ GItems[$u$]) then
        $j = SRSI(i,$ GroupItems$)$;
    end if
    if (getSelectProb() > $p^*$ & $i \in$ RItems[$u$]) then
        $j = SRSI(i,$ ItemSet$)$;
    end if
    $P_u = (P_u \setminus i) \cup j$;
end for
return: $P_u$;                                             ▷ AlterEgo profile for user u
```

Interestingly, D2P can also be applied in recommendation infrastructures where the KNN is computed by third-party cloud services that act as intermediaries between the recommendation server and users: these servers create the *AlterEgo* profiles, preserving privacy with respect to a server. Moreover, D2P can be applied by the users themselves (in P2P or hybrid infrastructures [27]), preserving privacy of users against other users.

C. KNN Selection Phase

In user-based CF recommenders, a K-Nearest Neighbors (KNN) [175] algorithm computes the K most similar users based on some similarity metric (Phase 2 in Algorithm 1). In this phase, we periodically update the top-K_{users} similar users for an active user as the $neighbors$ using the *AlterEgo* profiles generated in the modification phase.

D. Recommendation Phase

In this final phase, the recommendations are computed using those K_{users} neighbors. In the context of this work, we select the most popular items among the *neighbors* of u to be recommended to u (similar to HyRec in §3.1).

D2P requires some maintenance operations which are as follows.

- *Profile update:* When a user u rates an item i, then both P_u and P_i are updated. Profiles are updated incrementally as in standard online recommenders.
- *Group update:* The static nature of the relationship (similarity) [111, 157] between items stabilizes the grouping phase. So, the frequency of group updates has little impact on the quality of the provided recommendations; The groups are updated periodically after every 10 days in our evaluation.
- *Recommendation:* The new recommendations are delivered to the active user incrementally whenever an item is rated by the user. In D2P, only the *AlterEgo* profiles of the KNN are updated during each recommendation. We take into account the recent ratings provided by the users to compute recommendations.

E. D2P Privacy Analysis

We now analyze our D2P privacy in the recommender model introduced above.

First, we denote the $GroupItems$ for an item i in Algorithm 7 as:

$$\mathcal{G}_\lambda(i) = \left(\cup_{j \in \text{KNN}(\mathcal{GRP}_\lambda(i))} \mathcal{GRP}_\lambda(j)\right) \cup \mathcal{GRP}_\lambda(i)$$

As mentioned earlier, the selector selects to replace an element s with any random element from $\mathbb{E}$ with a probability p and with any random element from $\mathcal{G}_\lambda(s)$ with a probability $1-p$. So, it finally outputs two sets of elements *GItems* and *RItems* for each user profile. For both of these sets (*GItems* and *RItems*), the profiler retains the original elements with probability p^*. It replaces elements in *GItems* with elements from $\mathcal{G}_\lambda(s)$ and elements in *RItems* with any possible element $e \in \mathbb{E}$ with probability $1-p^*$. Here $\mathcal{N}_\mathbb{E}$ is the total number of elements in $\mathbb{E}$.

We now provide the following remark concerning the privacy parameter ϵ from Definition 13. (Further details about the following remark along with additional formal proofs for an in-depth privacy analysis are provided in [76] for interested readers.)

Remark 2 (PRIVACY QUANTIFICATION)**.** *For any given distance metric λ and any two elements i and j, we denote $\mathcal{SUB}(i,j)$ the event of substituting element i with j in any mechanism $\mathcal{M}$. This substitution probability is denoted by $Pr(\mathcal{SUB}(i,j))$. Then, for any mechanism $\mathcal{M}$, we have ϵ as:*

$$\epsilon = \ln\left(\max_{i,j,k \in \mathbb{E}\, and\, i \neq j}\left(\frac{Pr(\mathcal{SUB}(i,k))}{Pr(\mathcal{SUB}(j,k))}\right)\right)$$

We now compute the substitution probability for any two arbitrary elements s and t, in this abstract recommender model. We get the following:

$$Pr(\mathcal{SUB}(s,t)) = \begin{cases} p^* + \frac{(1-p)(1-p^*)}{|\mathcal{G}_\lambda(s)|} + \frac{p(1-p^*)}{\mathcal{N}_\mathbb{E}} & \text{if } s = t \\ \frac{(1-p)(1-p^*)}{|\mathcal{G}_\lambda(s)|} + \frac{p(1-p^*)}{\mathcal{N}_\mathbb{E}} & \text{if } t \in \mathcal{G}_\lambda(s) \setminus s \\ \frac{p(1-p^*)}{\mathcal{N}_\mathbb{E}} & \text{if } t \notin \mathcal{G}_\lambda(s) \text{ .} \end{cases}$$

Let $\epsilon_{\text{D2P}}^{(p,p^*,\lambda)}$ denote the ϵ for D2P with privacy parameters (p, p^* *and* λ) and $|\mathcal{G}_\lambda|$ denote $\min_{s \in E}(|\mathcal{G}_\lambda(s)|)$. Then, using the above substitution probabilities and Remark 2, we get:

$$\epsilon_{\text{D2P}}^{(p,p^*,\lambda)} = \ln(1 + \frac{p^* + \frac{(1-p)(1-p^*)}{|\mathcal{G}_\lambda|}}{\frac{p(1-p^*)}{\mathcal{N}_\mathbb{E}}}) \tag{5.3}$$

So, when we compute using the original profile, we have $p^* = 1$, which implies $\epsilon_{\text{D2P}}^{(p,1,\lambda)} = \infty$ (*no privacy*). When $p^* = 0$ in Equation 5.3, so all the items are replaced with some items. Then we have $\epsilon_{\text{D2P}}^{(p,0,\lambda)}$ as :

$$\epsilon_{\text{D2P}}^{(p,0,\lambda)} = \ln(1 + \frac{\frac{(1-p)}{|\mathcal{G}_\lambda|}}{\frac{p}{\mathcal{N}_\mathbb{E}}}) = \ln(1 + \frac{(1-p).\mathcal{N}_\mathbb{E}}{p.|\mathcal{G}_\lambda|}) \tag{5.4}$$

From this $\epsilon_{\text{D2P}}^{(p,0,\lambda)}$, we see that when p increases, the probability to replace an item with a random item increases leading to more privacy and that is evident from the decreasing value of $\epsilon_{\text{D2P}}^{(p,0,\lambda)}$ in Equation 5.4. When $p = 1$ in Equation 5.4, D2P achieves $\epsilon_{\text{D2P}}^{(1,0,\lambda)} = 0$ (*perfect privacy*). For larger λ, the size of the groups becomes larger, hence privacy increases resulting in smaller ϵ_{D2P}.

5.1.4 Evaluation

This section presents an exhaustive experimental evaluation of our D2P-based recommender using two real-world datasets namely Jester and MovieLens. In particular, we compare the recommendation quality and coverage [65] of D2P with that of a non-private protocol directly relying on the original user profiles. We also provide a comparison with [132], one of the closest to our work. Additionally, we discuss an item-based version of D2P (*i*-D2P) which we also implemented and evaluated.

A. Experimental Setup

Evaluation scheme. We measure the recommendation quality as follows: we divide the dataset into a training set (80% of the dataset trace) and a test set (20%). For each rating in the test set, a set of top recommendations is selected as the *Recommendation Set* (RS). We denote the size of the recommendation set as N. More precisely, we evaluate the extent to which the recommender is able to predict the content of the test set while having computed the KNN on the training set.

Evaluation metrics. We use *Precision* and *Recall* as our evaluation metrics (§2.7). To get an estimate of the drop in quality, we measure the decrease in precision for Top-5 recommendations [137] (denoted by Pr@5), as most recommenders follow Top-N recommendations, e.g: *IMDB* uses *Top-6* list to suggest similar movies, *Amazon* uses *Top-4* list to suggest similar products and *last.fm* uses *Top-5* list to suggest similar music.

Datasets. We evaluate D2P with two datasets: the MovieLens (ML) dataset [138] and the Jester one [96]. The ML dataset consists of $100,000$ (100K) ratings given by 943 users over 1682 movies. The Jester dataset [96] contains 4.1 million ratings of 100 jokes from 73,421 users. We use a subset of the Jester dataset with around 36K ratings given by 500 users over 100 jokes. The Jester subset consists of 500 users selected uniformly at random among all users who rated at least 50 jokes. D2P relies on the item-replacement technique, so the quality of the recommendation provided by D2P depends on how much two items are connected in the dataset. We thus consider datasets with diverse characteristics to evaluate D2P.
Diversity: We created 4 diverse datasets from the ML $100K$ dataset to cover a variety of characteristics (typically sparsity). The ratings are stored in a user-item matrix where the rows of the matrix contain the user-ids and the columns contain the item-ids. Then, the rows are sorted based on the total number of ratings given by the users and the columns are sorted based on the total number of times the items have been rated by different users. The partitioning of the dataset is shown in Figure 5.4 as *users* × *items* matrix.

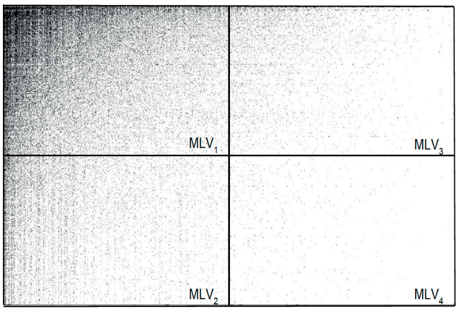


Figure 5.4 – *ML1 Dataset Partitions based on rating density.*

Characterization. To evaluate D2P in different settings, we characterize the datasets according to *rating density* metric. The rating density (RD) is the ratio of the number of ratings given by the users in the dataset to the total number of ratings possibly given (number of users multiplied by the number of items).

Table 5.1 depicts the rating densities of different datasets.

Dataset	#Users	#Items	Ratings	RD(%)
$Jester$	500	100	36000	71.01
$ML1$	940	1680	99647	6.31
MLV_1	470	840	76196	19.3
MLV_2	470	840	16187	4.1
MLV_3	470	840	6317	1.6
MLV_4	470	840	750	0.19

Table 5.1 – *Datasets characteristics.*

B. Impact of Rating Density

Figure 5.5 shows the recall measured with varying size of the recommendation set in D2P with parameters $p = 0.5$, $p^* = 0.5$ and $\lambda = 1$. We observe that higher rating density results in better recall using D2P. As shown in Table 5.1, the rating density of the MovieLens 100K dataset is 6.31% and that of its 4 subsets varies with a maximum of 19.3% and minimum of about 0.19%. From Figure 5.5, we observe that D2P is not suitable for datasets with too low rating densities, like MLV_3 and MLV_4, as these result in lower $recall$. However, we observe, for MLV_2, D2P provides slightly better recall compared to a more dense dataset (like MLV_1). This happens because the number of items relevant to a user (in the test set) is less in MLV_2 (more sparse) compared to MLV_1 (less sparse). However, for more sparse datasets like MLV_3 or MLV_4, collaborative filtering is not effective because the ratings are insufficient to identify similarities in user interests.

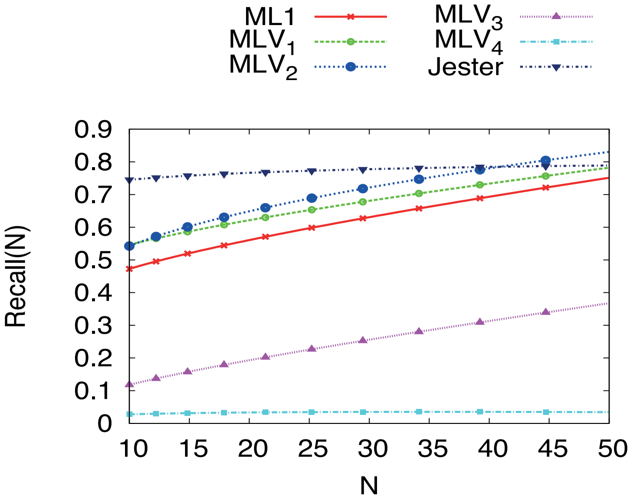
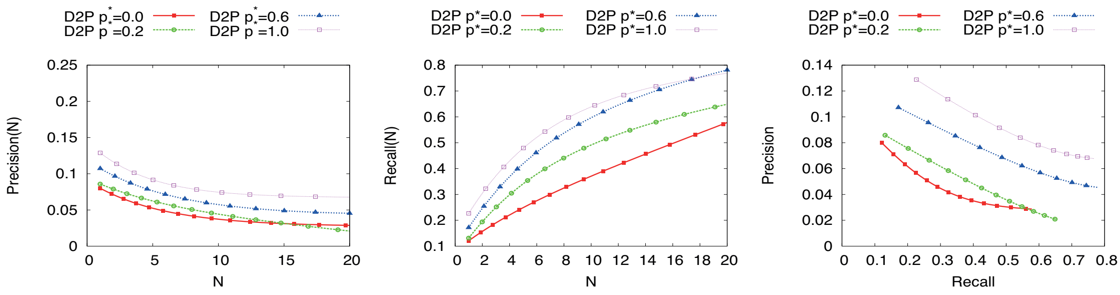


Figure 5.5 – *Recall@N with varying Dataset Characteristics.*

C. Privacy-Quality Trade-off

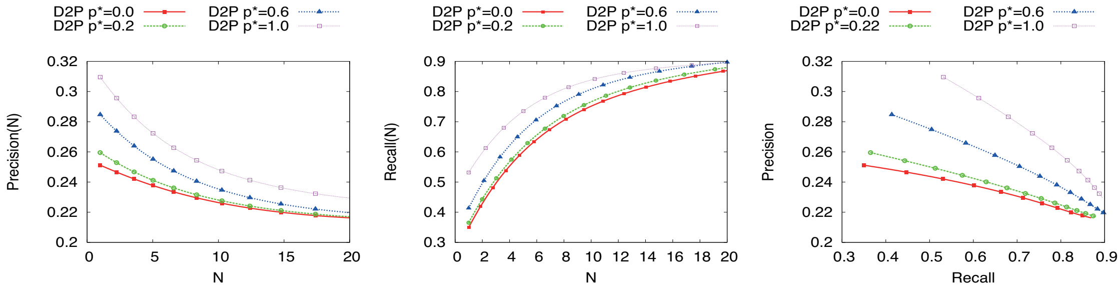
Effect of profiler probability (p^*). We vary the value of parameter p^* from the *Profiler* algorithm from a minimum of 0 to a maximum of 1 (*no privacy*) with other parameters $\lambda = 1$, $p = 0.5$.

MovieLens. Figure 5.6 demonstrates the performance of the D2P over several values of p^* on the MovieLens dataset. In Figure 5.6(a), we observe that the quality drops only by 3.24%, in terms of Pr@5, when compared to a non-private approach ($p^* = 1$).
Jester. Figure 5.7 shows the results of the performance of the D2P over several values of p^* on Jester workload. In Figure 5.7(a), we observe that the quality drops only by 2.9% in terms of Pr@5. Interestingly, we observe in Figure 5.7(b) that the recall of a non-private approach ($p^* = 1$) is very similar to the one achieved by D2P (e.g, at $N = 20$, the *recall* values differ by 0.02 only). This observation also means that D2P provides good recommendation quality in datasets with higher rating densities. The higher the profiler probability, the better the recommendation quality.



(a) *Precision@N Comparison.* **(b)** *Recall@N Comparison.* **(c)** *Precision-Recall Comparison.*

Figure 5.6 – *Effect of Profiler Probability (p^*) on Quality for the ML Dataset (User-based CF).*

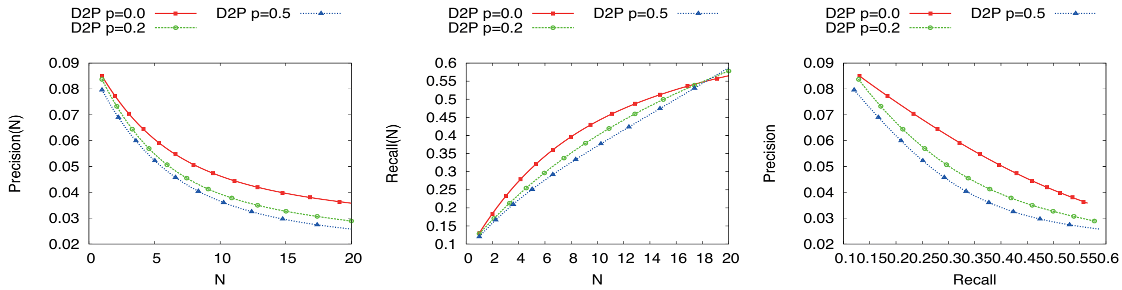


(a) *Precision@N Comparison.* **(b)** *Recall@N Comparison.* **(c)** *Precision-Recall Comparison.*

Figure 5.7 – *Effect of Profiler Probability (p^*) on Quality for the Jester Dataset (User-based CF).*

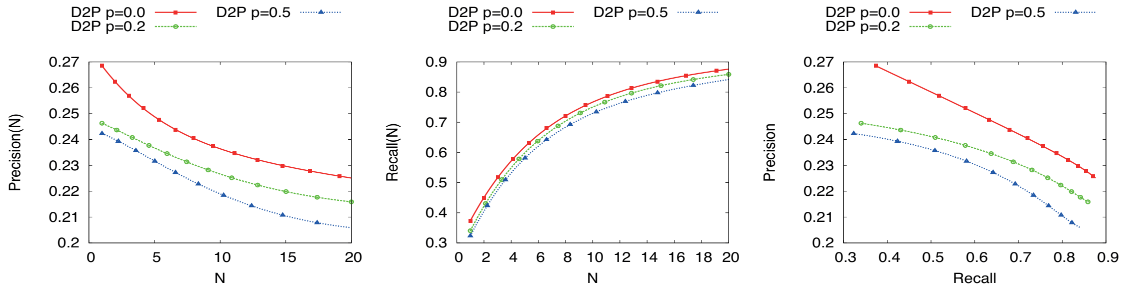
Effect of selector probability (p). Here, we vary the probability p from the *Selector* algorithm from a minimum of 0 to a maximum of 0.5 (with $\lambda = 1$, $p^* = 0$).
MovieLens. Figure 5.8 demonstrates the performance of D2P over several values of p on MovieLens.
Jester. Figure 5.9 shows the results of the performance of D2P over several values of p on Jester dataset. The lower the selector probability, the better the recommendation quality.

Effect of distance metric (λ). We also analyzed the effect of varying the level of privacy using the distance parameter: λ. We observed the quality of recommendations provided by D2P



(a) *Precision@N Comparison.* **(b)** *Recall@N Comparison.* **(c)** *Precision-Recall Comparison.*

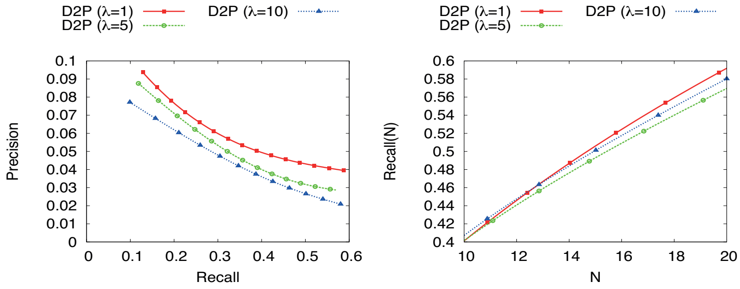
Figure 5.8 – *Effect of Selector Probability (p) on Quality for the ML Dataset (User-based CF).*



(a) *Precision@N Comparison.* **(b)** *Recall@N Comparison.* **(c)** *Precision-Recall Comparison.*

Figure 5.9 – *Effect of Selector Probability (p) on Quality for the Jester Dataset (User-based CF).*

with several values of λ (with $p = 0.5$, $p^* = 0$). The results of these experiments are given in Figure 5.10. We observe that a lower λ provides better quality because items gets replaced by closer items for lower λ.



(a) *Precision-Recall Comparison.* **(b)** *Recall@N Comparison.*

Figure 5.10 – *Effect of Distance Metric (λ) on Quality for the ML Dataset (User-based CF).*

D. Parameter Selection

The distance parameter λ is used to protect user's privacy. We now illustrate its usage on two examples. The first one is depicted in Figure 5.11. We consider 3 categories (A,B,C), 3 users (U_1, U_2, U_3) and 5 movies (I_1, I_2, I_3, I_4, I_5). We assume that each user wants to hide some specific category. To hide a Category A for user U_1, we anonymize it with at least one different Category (B or C). We can achieve this by computing the minimum distance for items from Category A in U_1's profile (I_1, I_3) to items in different categories. For item I_1, we get the distance is 2.8 to I_2 in Category B and 3 to I_4 in Category C. So, the minimum distance for I_1 is 2.8 to I_2 in Category B. We get the same for I_3 in U_1's profile. Now, to satisfy the distance for both of these items, we choose the maximum among them which is 2.8. This gives us the λ_{U_1} to hide Category A for U_1. We do the same for users U_2 and U_3. Finally, to set the λ for the system, we get the maximum from all users (which is 2.8 in the example).

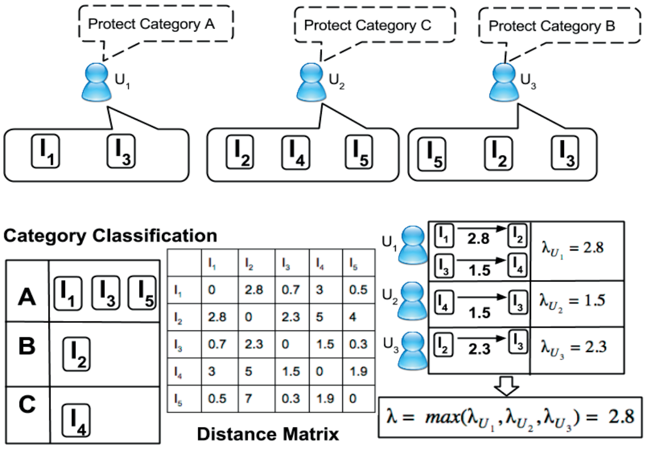


Figure 5.11 – *Distance for Personal Choice.*

The distance parameter can be also selected as the average distance for each user profile (λ_k). Here, λ_k for user U_k is computed as the average value of the distance between all pairs of items rated by user U_k. Figure 5.12 provides an intuition for this distance parameter. For the datasets used for evaluation, we get $\lambda_{ML1} = 6.5$, $\lambda_{Jester} = 1.5$.

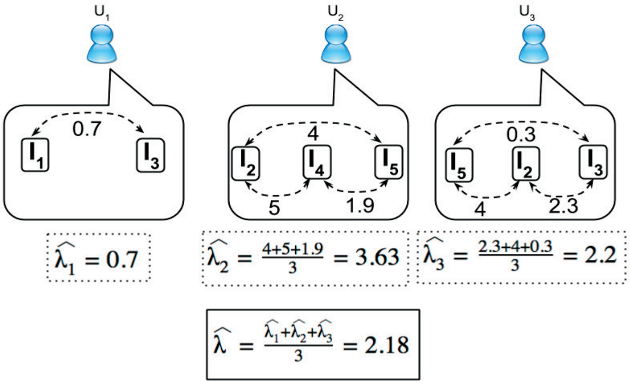
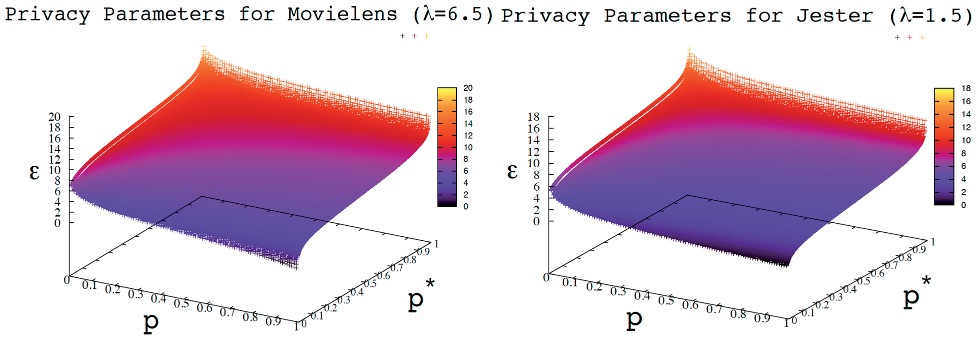

U1
U2
U3
0.7
I1 I3
4
I2 I4 I5
5 1.9
0.3
I5 I2 I3
4 2.3
$\widehat{\lambda_1} = 0.7$
$\widehat{\lambda_2} = \frac{4+5+1.9}{3} = 3.63$
$\widehat{\lambda_3} = \frac{2.3+4+0.3}{3} = 2.2$
$\widehat{\lambda} = \frac{\widehat{\lambda_1}+\widehat{\lambda_2}+\widehat{\lambda_3}}{3} = 2.18$

Figure 5.12 – *Distance for Average.*

To demonstrate the degradation of ϵ based on parameters, p and p^*, we fix the distance parameter (λ_{ML1} = 6.5, λ_{Jester} = 1.5). Figure 5.13 demonstrates the degradation of ϵ based on the privacy parameters (p, p^*). For MovieLens, we obtain good privacy (ϵ= 2.9) and good quality (F_1-score=8.5%) with p =0.7, p^*=0.03, λ=6.5. For Jester, we obtain good privacy (ϵ= 0.97) and good quality (F_1-score=23.1%) with p =0.8, p^*=0.01, λ=1.5.



(a) *Privacy Parameters for MovieLens (ML1).* **(b)** *Privacy Parameters for Jester.*

Figure 5.13 – *Privacy Parameters Comparison.*

E. Coverage Evaluation

Beyond accuracy, there is a variety of other metrics that should be used to evaluate a recommender [65, 85]. The *Coverage* of a recommender is a metric that captures the domain of items over which it can make recommendations. In particular, we evaluate *Catalog Coverage* [65] of D2P and compare it to the coverage provided by a standard non-private recommender. Consider a set of items I_K^j contained in the *Top-K* list during the j^{th} recommendation instance. Also, denote the total number of items by N. Hence, *Catalog Coverage* after M recommendation instances can be mathematically represented as follows:

$$Catalog\ Coverage = \frac{|\cup_{j=1...M} I_K^j|}{N}$$

Figure 5.14 demonstrates the *Catalog Coverage* for D2P and compares it with the coverage in a standard recommender for MovieLens. We observe that D2P provides 1.5 times better coverage than a standard recommender when the size of recommendation set is 1.

F. Overhead Evaluation

We evaluate here the computational overhead of D2P's privacy and compare it to the one of [132] which we denote as DP_δ. We call the computations performed for every recommendation as *Online* computations and the computations done periodically as *Offline* computations. We compare the privacy overhead with the Recommendation Latency (RL) in D2P. Additionally, we compare the privacy overhead in D2P with the privacy overhead in DP_δ. As shown in Table 5.2, the overhead for the offline computations in D2P is around 26.4 times smaller than that of [132] for MovieLens and around 4.5 times smaller for Jester. All

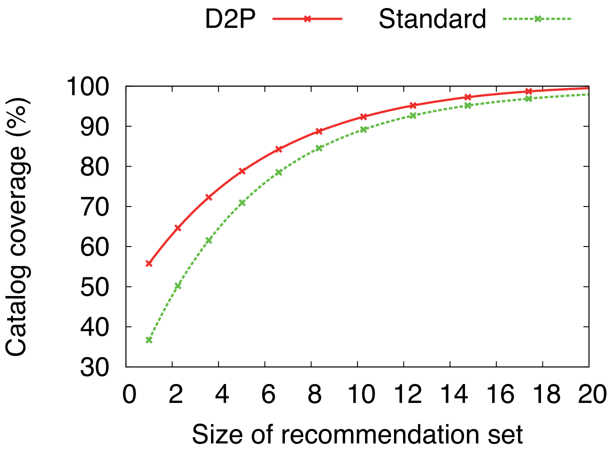


Figure 5.14 – *Catalog Coverage Comparison.*

offline computations are parallelised on a 8-core machine.

Datasets	D2P Overhead			DP_δ Overhead
	RL	Online	Offline	Offline
$ML1$	196ms	32ms	4.54s	120s
$Jester$	24ms	12ms	162ms	740ms

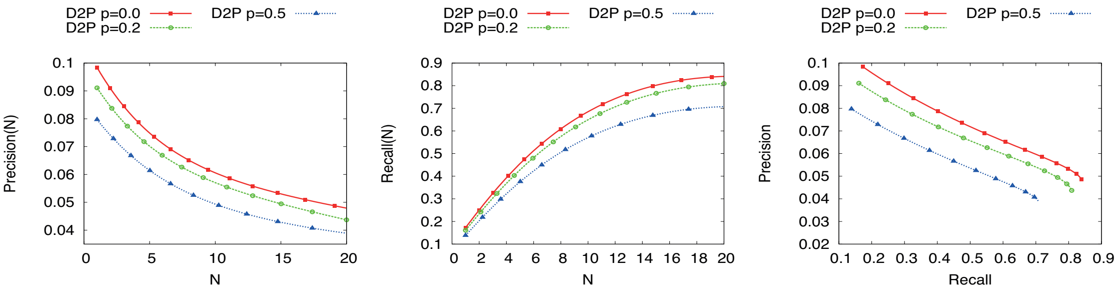
Table 5.2 – *Overhead of Privacy.*

G. Item-based D2P

D2P can be used with any collaborative filtering technique. We evaluate D2P in another context to illustrate the genericity of D2P. We implemented an item-based version of D2P: i-D2P. In i-D2P, the grouping phase is responsible for creating groups of similar *users* based on the distance metric λ. The selector and profiler components in i-D2P create *AlterReplica* profiles of the items using the same approach as in D2P. Finally, the item recommendations are computed using these *AlterReplica* profiles during the recommendation phase in i-D2P. Figure 5.15 conveys the quality of recommendations provided by i-D2P for varying values of parameter p (with $\lambda = 1$, $p^* = 0$). Figure 5.16 conveys the quality of recommendations provided by i-D2P for several values of parameter p^* (with $\lambda = 1$, $p = 0.5$). In Figure 5.16(a), we observe that the quality drops by 1.89% in terms of Pr@5 for the ML dataset. This shows that D2P also provides good quality of recommendations in item-based CF recommenders.

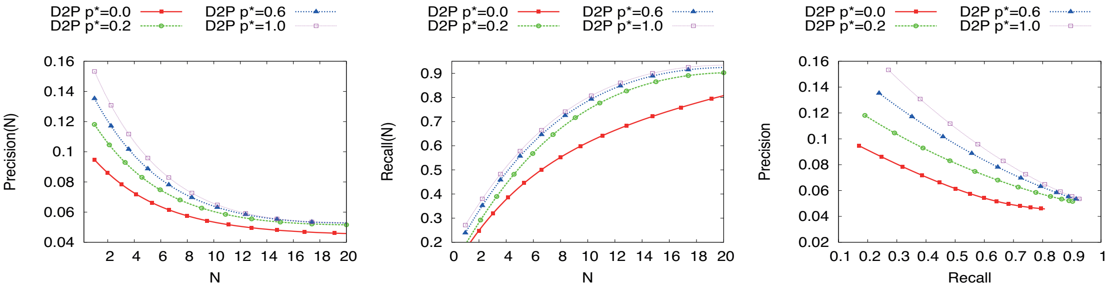
5.1.5 Related Work

The notion of differential privacy was introduced by Cynthia Dwork [55, 57, 64]. Most of the research focused on theoretical aspects and provided feasibility and infeasibility results [100]. In this work, we extend differential privacy to the context of recommenders. We appended



(a) *Precision@N Comparison.* **(b)** *Recall@N Comparison.* **(c)** *Precision-Recall Comparison.*

Figure 5.15 – *Effect of Selector Probability (p) on Quality for the ML Dataset (Item-based CF).*



(a) *Precision@N Comparison.* **(b)** *Recall@N Comparison.* **(c)** *Precision-Recall Comparison.*

Figure 5.16 – *Effect of Profiler Probability (p^*) on Quality for the ML Dataset (Item-based CF).*

the original definition with a distance metric (λ) and presented an effective way to achieve it through our D2P protocol.

Polat et. al. [146] proposed a randomized perturbation technique to protect user's privacy. Zhang et. al. [194] showed however that a considerable amount of information can be derived from randomly perturbed ratings. Instead of adding perturbations to user profiles, D2P uses the *AlterEgo* profiles which are created based on a distance threshold (λ). Privacy breaches (compromised user identities) occur when e-commerce sites release their databases to third-parties for data-mining or statistical reporting [1]. The fact that with D2P, the third-parties have only access to the *AlterEgo* profiles alleviates the risk of revealing user's identity to those third parties.

In fact, although, there had been a lot of research work related to privacy in online recommenders [102, 127] and differential privacy [55, 57, 64], only a few of these combined these two notions [93, 132]. McSherry et. al. designed a relaxed version of differential privacy in the context of recommenders [132]. In short, the idea is to add to the ratings - a limited amount of - Gaussian noise. Our notion of distance-based differential privacy provides a stronger form of classical differential privacy in the context of recommender systems. In our case, we replaced items in users profiles with others at some distance. Other differences between the two approaches include the way dynamic updates are addressed as well as the underlying overhead. McSherry et. al. does not consider updates to the covariance matrix, and hence is not applicable to a dynamic system without jeopardizing the privacy guarantee. The *AlterEgo* profiles used in D2P can grow naturally without the need to recompute from scratch like in [132]. Also, the underlying overhead in D2P is lower. As shown in Table 4, the overhead in D2P is around 26.4 times smaller than that of [132] for MovieLens and around 4.5 times smaller for Jester. The additional overhead in [132] stems from the compute-intensive preprocessing steps: (i) removal of per-movie global effects and (ii) centering and clamping process.

5.1.6 Conclusion

While personalization has become crucial on the web, it raises however privacy concerns as its quality relies on leveraging user profiles. In this work, we present an extension of the notion of differential privacy to the context of recommenders: systems that personalize recommendations based on similarities between users. We introduced D2P which ensures this strong form of privacy. D2P addresses the trade-off between privacy and quality of recommendation: it can be applied to any collaborative recommender.

The main intuition behind D2P is to rely on a distance metric between items so that groups of similar items can be identified. D2P leverages this notion of group to generate, from real user profiles, alternative ones, called *AlterEgo* profiles. These represent differentially private versions of the exact profiles. Such profiles are then used to compute the KNN and provide recommendations. We analyze D2P and evaluate experimentally the impact of the privacy mechanism on the quality of the recommendation in the context of two datasets: MovieLens and Jester. Our results show that privacy can be ensured without significantly impacting the quality of the recommendation. Our experiments demonstrate that D2P can provide 1.5 times better coverage than a standard recommender for MovieLens. Additionally, D2P incurs a small privatization overhead compared to other privacy-preserving system like [132] which makes it comparatively more practical for dealing with real-time workloads. D2P could be further extended to other filtering techniques that rely on user profiles for their recommendation computations. It is also possible to incorporate a hybrid approach in D2P where the item groups would be formed using content-based filtering [182] while the actual recommendations would be made based on collaborative filtering techniques.

One limitation of D2P stems from the fact that the users trust the service-providers with the original user profiles. Privacy could hence be compromised by online spying on users' activities [71]. It would be interesting to study the impact on privacy and recommendation quality of probabilistically altering or encrypting user rating [4]: the goal would be to preserve the profile anonymity even from service-providers. Combining such techniques with D2P would result in a recommender which is robust to malicious users (*user-level privacy*) and even untrusted service-providers engaged in spying activities (*system-level privacy*).

5.2 System-level Privacy

Recall that a service provider collects data from users in the form of *profiles* to compute neighbors and recommendations. This, however, opens major *system-level privacy* concerns in the sense that the profile of any user (say Alice) might get leaked from service providers [156]. E-commerce sites often release their databases to third-parties for data mining, intrusion detection and statistical reporting [150].

We designed X-REC, a novel recommender which ensures the privacy of users against the service providers (*system-level privacy*) or other users (*user-level privacy*) with negligible increase of latency in providing recommendations to end-users, while preserving recommendation quality. X-REC builds over two underlying services: a homomorphic encryption scheme over integers to encrypt user profiles, called X-HE, and a neighborhood selection protocol over the encrypted profiles, called X-NN. We provide efficient implementations of both these services. X-NN operates over data encrypted under X-HE and selects nearest neighbors if their similarities pass a given similarity threshold (T). It emulates the truth tables of the two logical gates XOR and AND with integer operations and thus circumvents the necessity of FHE. We employ a *uniform user sampling* technique which, we show, guarantees differential privacy [56] in the context of a recommender. Unlike in recent privacy-preserving systems [18, 33, 87, 62] where users are required to be logged-in, X-REC does not restrict the *dynamicity* [4] of the system. For interested readers, a more detailed information regarding how X-REC provides system-level privacy is provided in the following very interesting work [77].

---

[4] Users can *log-in/log-out* (resp. *join/leave*) at any time.

PART IV

Heterogeneity

As of today, most recommenders are *homogeneous* in the sense that they utilize one specific application at a time. In short, Alice will only get recommended a movie if she has been rating movies. But what if she would like to get recommendations for a book even though she rated only movies? Clearly, the multiplicity of domains (movies, books, songs) is calling for *heterogeneous* recommenders that could utilize ratings for one domain to provide recommendations in another one. This chapter of the thesis presents novel heterogeneous recommenders based on the preferences of users across various domains.

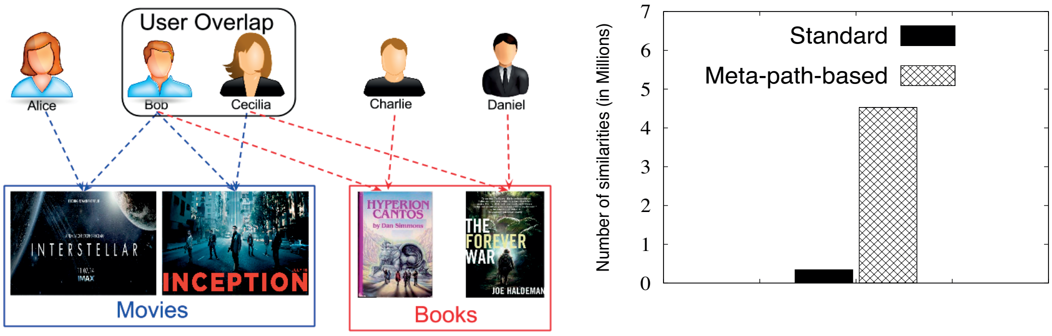
- We first present a heterogeneous recommender system (X-MAP) which enables recommendations across multiple domains based on user-item interactions (e.g., ratings) in §6.1.
- We also briefly explore the possibility of content-enabled heterogeneous recommendations in §6.2.

6 Heterogeneous Recommendations

6.1 Heterogeneous Recommendations with Alter-Egos

6.1.1 Overview

The next level to personalization is *heterogeneity*, namely *personalization across multiple domains* [44]. Heterogeneous preferences on the web, i.e., preferences from multiple application domains, should be leveraged to improve personalization, not only for users who are new to a given domain (i.e, *cold-start* situation), but also when the data is *sparse* [2] (e.g, very few ratings per user). In fact, if a user, say Alice, likes the *Interstellar* movie, then a heterogeneous personalization scheme could actually recommend her books such as *The Forever War* by Joe Haldeman. To get an intuition of how such recommendation can be made by going beyond standard schemes, consider the scenario depicted in Figure 6.1(a) where five users rated *at most* one book. Indeed, according to a standard metric (*adjusted cosine* [157]), the similarity between *Interstellar* and *The Forever War* is 0, for there are no common users who rated both. However, a closer look reveals the following *meta-path* [1] between these two heterogeneous items: *Interstellar* $\xrightarrow{Bob}$ *Inception* $\xrightarrow{Cecilia}$ *The Forever War*.



(a) *A simple scenario depicting heterogeneity across two domains.*

(b) *The effect of* meta-paths *in computing heterogeneous similarities.*

Figure 6.1 – *Heterogeneous recommendation using meta-paths.*

---

[1] We call *meta-path* any path involving heterogeneous items, e.g., movies and books.

Figure 6.1(b) compares the number of heterogeneous similarities that could be exhibited with or without using meta-paths on real-world traces from Amazon (using two domains: movies and books). Meta-path-based heterogeneous similarities clearly lead to better recommendation quality as we show later in §6.1.6.

A. Challenges

While appealing, building a practical heterogeneous meta-path-based recommender raises several technical challenges.

Meta-path-based similarity. Consider an undirected graph G where the vertices represent the items and each edge e_{ij} is associated with a weight s_{ij}, representing the similarity between items i and j. A meta-path in G can be defined as a sequence of adjacent vertices (movies or books) connected by edges in G. Computing a *heterogeneous* similarity based on these meta-paths is, however, not straight-forward. Such similarity could be affected by factors like the number of users involved, directly or indirectly (in the meta-paths), as well as the strength of the ties between item-pairs connected by (shorter) meta-paths. The challenge here is to capture these factors in a way that improves the accuracy of heterogeneous similarities.

Scalability. Clearly, the computational complexity increases many-fold while computing meta-path-based similarities. Computing all possible meta-paths on a large-scale graph with millions of vertices (heterogeneous items) can quickly become computationally intractable.

Privacy. Heterogeneous recommendations also raise privacy concerns. For example, the new transitive link between Alice and Cecilia (Figure 6.1(a)) provides the opportunity for a curious user, say Alice, to discover the *straddlers*: people like Bob or Cecilia who connect multiple domains. Alice can actually determine the item(s) that allows her to get this recommendation by pretending to be another user and incrementally rating items until she gets the recommendation. This is similar to the privacy risk in statistical database queries where inferences can be derived from combinations of queries [149]. As pointed out in [150], such straddlers are at a privacy risk, and information about their preferences could be used in conjunction with other data sources to uncover identities and reveal personal details. This can be particularly problematic across different applications like Netflix (movies) and Last.fm (music).

Recent heterogeneous recommenders [164, 44], extending classical homogeneous recommendation schemes across domains, are neither scalable nor private, and hence are not suitable for applications involving millions of users and items.

B. Contributions

In this work, we present a recommender we call X-MAP: **Cross**-do**ma**in **p**ersonalization system. X-MAP fully utilizes the overlap among users across multiple domains, as depicted in Figure 6.1(a). This overlap is often derived from profiles maintained by users across various web applications along with interconnection mechanisms for cross-system interoperability [36] and cross-system user identification [35]. At the heart of X-MAP lie several novel ideas.

- We introduce a novel similarity metric, X-SIM, which computes a meta-path-based transitive closure of inter-item similarities across several domains. X-SIM involves adaptations, to the heterogeneous case, of classical *significance weighting* [84] (to account for the number of users involved in a meta-path) and *path length* [150] (to capture the effect of meta-path lengths) schemes.
- We introduce the notion of *AlterEgos*, namely artificial profiles (created using X-SIM), of users even in domains where they have *no* or *very little* activity yet. We generate an AlterEgo profile (of Alice) in a target domain leveraging an item-to-item mapping from a source domain (e.g., movies) to the target domain (e.g., books). AlterEgos enable to integrate any standard recommendation feature in the target domain and preserve, for example, the temporal behavior of users [53].
- We use an effective layer-based pruning technique for selecting meta-paths. AlterEgos, acting as a caching mechanism, alleviate computational intractability by only using the information from the target domain. Combined with our layer-based pruning technique, AlterEgos enable X-MAP to scale almost linearly with the number of machines (a major requirement for the deployment of a recommender in a practical environment). We illustrate this scalability through our implementation of X-MAP on top of Apache Spark [189].
- We introduce an obfuscation mechanism, based on meta-path-based similarities, to guarantee differentially private AlterEgos. We adapt, in addition, a probabilistic technique, inspired by Zhu et al. [199, 200], to protect the privacy of users in the target domain. Interestingly, we show that, despite these privacy techniques, X-MAP outperforms the recommendation accuracy of alternative non-private heterogeneous approaches [14, 20, 44].
- We deployed an online recommendation platform, using X-SIM on a database of 660K items, to recommend books and movies to users based on their search queries at:

  http://x-map.work/

  Books like *The Da Vinci Code* are indeed recommended when the search query is the *Angels & Demons* (2009) movie. Currently, we support Chrome, Safari and Firefox browsers.

6.1.2 Heterogeneous Recommendation Problem

Without loss of generality, we formulate the problem using two domains, referred to as the *source* domain ($\mathcal{D}^S$) and the *target* domain ($\mathcal{D}^T$). We use superscript notations ^{S} and ^{T} to differentiate the source and the target domains. We assume that users in $\mathcal{U}^S$ and $\mathcal{U}^T$ overlap, but $\mathcal{I}^S$ and $\mathcal{I}^T$ have no common items. This captures the most common heterogeneous personalization scenario in e-commerce companies such as Amazon or eBay nowadays. The heterogeneous recommendation problem can then be stated as follows.

Problem 1. *Given any source domain $\mathcal{D}^S$ and any target domain $\mathcal{D}^T$, the heterogeneous rec-*

ommendation problem consists in recommending items in $\mathcal{I}^T$ *to users in* $\mathcal{U}^S$ *based on the preferences of* $\mathcal{U}^S$ *for* $\mathcal{I}^S$ *(ratings in the source domain),* $\mathcal{U}^T$ *for* $\mathcal{I}^T$ *(ratings in the target domain) and* $\mathcal{U}^S \cap \mathcal{U}^T$ *for* $\mathcal{I}^S \cup \mathcal{I}^T$ *(overlapping ratings).*

In other words, we aim to recommend items in $\mathcal{I}^T$ to a user who rated a few items (sparsity) or no items (cold-start) in $\mathcal{I}^T$. Figure 6.1(a) conveys the scenario that illustrates this problem. The goal is to recommend new relevant items from $\mathcal{D}^T$ (e.g., books) either to Alice who never rated any book (cold-start) or to Bob who rated only a single book (sparsity). Both the users rated items in $\mathcal{D}^S$ (e.g., movies).

6.1.3 X-SIM: Cross-domain similarity

We now present X-SIM, our novel similarity metric designed for heterogeneous recommendation along with our meta-path pruning technique.

A. Baseline Similarity Graph

We first build a baseline similarity graph where the vertices are the items and the edges are weighted by the similarities. We could use here any classical item-item similarity metric like Cosine, Pearson, or Adjusted-cosine [157] for baseline similarity computations. We choose to use adjusted-cosine for it is considered the most effective [157]:

$$s_{ac}(i,j) = \frac{\sum_{u \in \mathcal{U}_i \cap \mathcal{U}_j} (r_{u,i} - \bar{r_u})(r_{u,j} - \bar{r_u})}{\sqrt{\sum_{u \in \mathcal{U}_i} (r_{u,i} - \bar{r_u})^2} \sqrt{\sum_{u \in \mathcal{U}_j} (r_{u,j} - \bar{r_u})^2}} \tag{6.1}$$

In this first step, we compute the (baseline) similarities by integrating both $\mathcal{D}^S$ and $\mathcal{D}^T$ as a single domain. We denote by G_{ac} [2] the resulting similarity graph in which any two items are *connected* if they have common users. As shown in Figure 6.1(b), the limitation of adjusted-cosine similarity leads to sparse connections in G_{ac}. We address this sparsity issue of G_{ac} precisely by extending it with *meta-paths* connecting both domains.

Clearly, a brute-force scheme considering all possible meta-paths would be computationally intractable and not scalable. Assuming m items in the database, the time complexity of such a brute-force scheme (computing similarity for every pair of items) would be $O(m^2)$, which is not suitable for big datasets like the Amazon one with millions of items. X-MAP uses a layer-based technique to prune the number of meta-paths, thereby leading to $O(km) \simeq O(m)$ time complexity where $k \lll m$.

---

[2] Here *ac* denotes adjusted cosine.

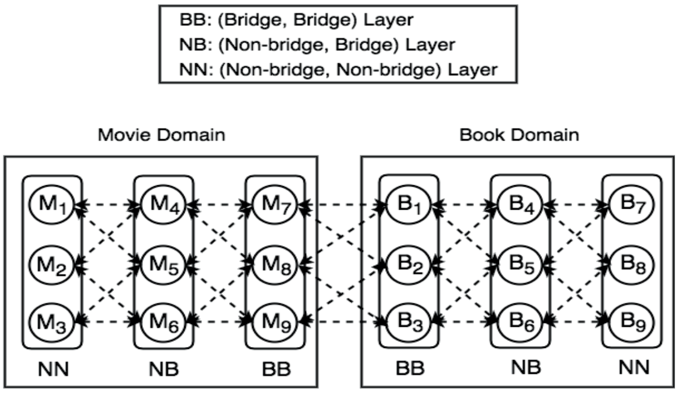

BB: (Bridge, Bridge) Layer
NB: (Non-bridge, Bridge) Layer
NN: (Non-bridge, Non-bridge) Layer
Movie Domain
Book Domain
M1
M2
M3
M4
M5
M6
M7
M8
M9
B1
B2
B3
B4
B5
B6
B7
B8
B9
NN
NB
BB
BB
NB
NN

Figure 6.2 – *Layer-based pruning in* X-MAP.

B. Layer-based Pruning

Based on the baseline similarity graph, we determine what we call *bridge items*, namely any item i in a domain $\mathcal{D}$ which connects to some item j in another domain $\mathcal{D}'$. Both i and j are bridge items in this case. These bridge items are ascertained based on the overlapping users from both domains. We accordingly call any item that is not a bridge item a *non-bridge item*.

X-MAP's pruning technique partitions the items from $\mathcal{D}^S$ and $\mathcal{D}^T$ into six possible layers, based on their connections with other items, as we explain below. In turn, the items in each domain, say $\mathcal{D}$, are divided into three layers (Figure 6.2).

- *BB-layer.* The (Bridge, Bridge)-layer consists of the bridge items of $\mathcal{D}$ connected to the bridge items of another domain.
- *NB-layer.* The (Non-bridge, Bridge)-layer consists of the non-bridge items of $\mathcal{D}$ which are connected to bridge items of $\mathcal{D}$.
- *NN-layer.* The (Non-bridge, Non-bridge)-layer consists of the non-bridge items of $\mathcal{D}$ which are not connected to other bridge items.

X-MAP then considers only the paths crossing different layers, which we call *meta-paths*. Since we use a k-nearest neighbor method in X-MAP, each item i in layer l is connected to the top-k items from every neighboring layer l' based on the item-item similarities. We describe our layered meta-path selection in more details in §6.1.5.

C. X-SIM: A Novel Similarity Metric

Consider any two items i and j. We denote by $\mathcal{U}_{i \geq \bar{i}}$ the set of users who rated item i higher than or equal to the average rating for i over all the users in the database who rated i. We also denote by $\mathcal{U}_{i < \bar{i}}$ as the set of users who rated item i lower than the average rating for i. Additionally, we denote by $|\mathcal{U}_i|$ the cardinality of the set $\mathcal{U}_i$.

Definition 14 (WEIGHTED SIGNIFICANCE). *Given any pair of items i and j, we define weighted significance ($S_{i,j}$) as the number of users who mutually like or dislike this given pair. Formally,* *we define the weighted significance ($S_{i,j}$) between i and j as follows.*

$$S_{i,j} = \underbrace{\left|\mathcal{U}_{i \geq \bar{i}} \cap \mathcal{U}_{j \geq \bar{j}}\right|}_{\textit{Mutual like}} + \underbrace{\left|\mathcal{U}_{i < \bar{i}} \cap \mathcal{U}_{j < \bar{j}}\right|}_{\textit{Mutual dislike}}$$

Intuitively, a higher significance value implies higher importance of the similarity value. For example, a similarity value of 0.5 between an item-pair (i, j) with $S_{i,j} = 1000$ is more significant than a similarity value of 0.5 between an item-pair (i,k) with $S_{i,k} = 1$ (for the latter may be a result of pure coincidence). [3]

Definition 15 (META-PATH)**.** *Given G and its six corresponding layers of items, a meta-path consists of at most one item from each layer.*

For every meta-path $p = i_1 \leftrightarrow i_2 \ldots \leftrightarrow i_k$, we compute the meta-path-based similarity s_p, weighted by its significance value, as follows.

$$s_p = \frac{\sum_{t=1}^{t=k-1} S_{i_t,i_{t+1}} \cdot s_{ac}(i_t, i_{t+1})}{\sum_{t=1}^{t=k-1} S_{i_t,i_{t+1}}}$$

For each pair of items (i, j) from different domains, if i, j are not connected directly, we aggregate the path similarities of all meta-paths between i and j. Due to the different lengths and similarities for meta-paths, we give different weights to different meta-paths. Shorter meta-paths produce better similarities in recommenders [150, 176] and hence are preferred over longer ones. We now explain the scheme behind assigning these weights and thereby computing the X-SIM values.

Definition 16 (NORMALIZED WEIGHTED SIGNIFICANCE)**.** *Given any pair of items i and j, we define normalized weighted significance ($\widehat{S}_{i,j}$) between i and j as their significance value weighted by the inverse of number of users rating either i or j. Formally, we denote normalized weighted significance as follows.*

$$\widehat{S}_{i,j} = \frac{S_{i,j}}{|\mathcal{U}_i \cup \mathcal{U}_j|}$$

Next, we determine the notion of *path certainty* (c_p) of a meta-path to take into account the factor of varying path lengths. Path certainty measures how good a path is for the similarity computations.

Definition 17 (PATH CERTAINTY)**.** *Given any meta-path ($p = i_1 \leftrightarrow i_2 \ldots \leftrightarrow i_k$), we compute the path certainty (c_p) of the meta-path p as the product of the normalized weighted significance between each consecutive pair of items in the path p. Formally, we define the path certainty as follows.*

$$c_p = \prod_{t=1}^{t=k-1} \widehat{S}_{i_t,i_{t+1}}$$

---

[3] This concept is analogous to statistical significance used in hypothesis testing.

It is important to note that the product of the normalized weighted significance values inherently incorporates the path length in our path certainty metric. Hence, shorter paths have higher weights compared to longer ones. Finally, we define our X-SIM metric as follows.

Definition 18 (X-SIM)**.** *Let $P(i,j)$ denote the set of all meta-paths between items i and j. We define the* X-SIM *for the item pair (i,j) as the path similarity weighted by the path certainty for all paths in $P(i,j)$. Formally, we define* X-SIM *for any given pair of items i and j as follows.*

$$\text{X-SIM}(i,j) = \frac{\sum\limits_{p \in P(i,j)} c_p \cdot s_p}{\sum\limits_{p \in P(i,j)} c_p}$$

Here, X-SIM(i,j) denotes the meta-path-based heterogeneous similarity between any two items i and j. X-SIM is then utilized to build the artificial profiles for users (*AlterEgos*). Note that a trivial transitive closure over similarities would not take into account the above-mentioned factors, which would in turn impact the heterogeneous similarities and consequently the recommendation quality.

6.1.4 X-MAP: Cross-domain recommender

We now show how to leverage our X-SIM metric to generate artificial (*AlterEgo*) profiles of users in domains where these users might not have any activity yet. For pedagogical reasons, we first present the non-private (NX-MAP) scheme, and then the extensions needed for the private (X-MAP) one.

A. Similarity Computation Phase

In this phase, X-MAP treats both the source and target domains as a single aggregated domain in order to compute pairwise item similarities, called *baseline* similarities. Basically, X-MAP computes the adjusted cosine similarities between the items in $\mathcal{I}^S \cup \mathcal{I}^T$ based on the preferences of the users in $\mathcal{U}^S \cup \mathcal{U}^T$ for these items. We distinguish the following two types of similarities:

(a) *Homogeneous similarities* are computed between items in the same domain. Such similarities are used for intra-domain extensions in §6.1.5.
(b) *Heterogeneous similarities* are computed between items in different domains. Such similarities are used for cross-domain extensions in §6.1.5.

B. X-SIM Computation Phase

After the computation of the baseline item-item similarities, X-MAP uses the X-SIM metric within a single domain to extend the connections between the bridge items of a domain and other items within the same domain. Then, X-MAP uses the X-SIM metric to extend the similarities between items across domains (we come back to this in more details in §6.1.5). After the heterogeneous similarity extension, each item in source domain ($\mathcal{D}^S$) has a corresponding set of items in target domain ($\mathcal{D}^T$) with quantified (positive or negative) X-SIM values.

C. AlterEgo Generation Phase

In this phase, the profile of Alice (in $\mathcal{D}^S$) is mapped to her AlterEgo profile (in $\mathcal{D}^T$) as shown in Figure 6.3. We first present the non-private case, and then discuss the private one.

NX-MAP AlterEgo generation. The non-private mapping is performed in two steps.

Replacement selection. In this step, for every item i in $\mathcal{D}^S$, we determine the replacement item j in $\mathcal{D}^T$. Here, j is the heterogeneous item which is most similar to i based on the heterogeneous similarities computed using X-SIM.

AlterEgo profile construction. We then replace every item rated by Alice in $\mathcal{D}^S$ with the most similar item in $\mathcal{D}^T$ computed in the previous step. This item replacement induces the AlterEgo profile [4] of Alice in the target domain as shown in Figure 6.3.

This AlterEgo profile of Alice is the mapped profile of Alice from the source domain to the target domain. Note that the AlterEgo profiles could be incrementally updated to avoid re-computations in X-MAP.

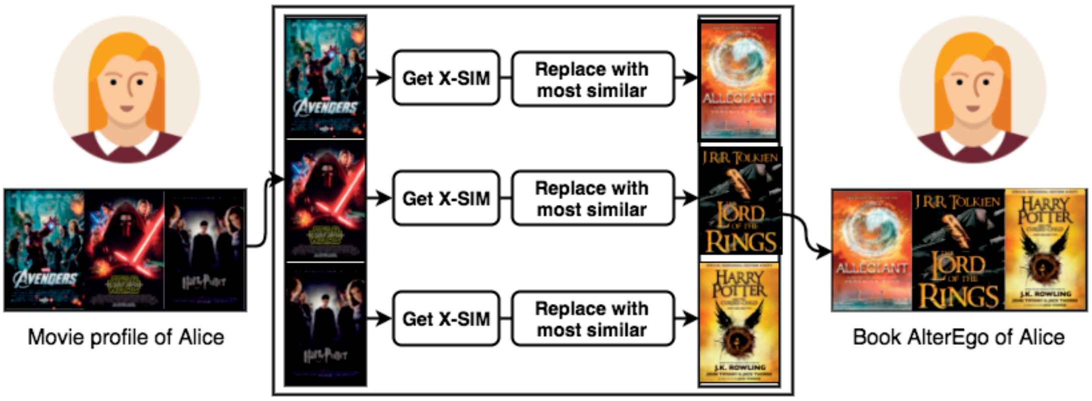

Get X-SIM
Replace with most similar
Get X-SIM
Replace with most similar
Get X-SIM
Replace with most similar
Movie profile of Alice
Book AlterEgo of Alice

Figure 6.3 – *Alice's AlterEgo profile (in target domain) mapped from her original profile (in source domain).*

X-MAP AlterEgo generation. We now explain how we achieve the differentially private mapping.

Private replacement selection. We apply an obfuscation mechanism, depending on the meta-path-based heterogeneous similarities, to design our differentially private replacement selection technique (Algorithm 8). Note that standard differentially private techniques consisting,

---

[4] If Alice has rated a few items in $\mathcal{D}^T$, then the mapped profile is appended to her original profile in $\mathcal{D}^T$ to build her AlterEgo profile.

for example, in adding noise based on Laplace or Gaussian distributions would not work here for they would not build a profile consisting of items in the target domain. The following theorem conveys our resulting privacy guarantee.

Theorem 4 (PRIVACY GUARANTEE)**.** *Given any item i, we denote the global sensitivity of* X-SIM *by GS and the similarity between i and any arbitrary item j by* X-SIM(i, j)*. Our Private Replacement Selection (PRS) mechanism, which outputs j as the replacement with a probability proportional to $exp(\frac{\epsilon \cdot \text{X-SIM}(i,j)}{2 \cdot GS})$, ensures ϵ-differential privacy.*

Proof. (The full proof is provided in Appendix §8.2.1 for interested readers.) Consider any two datasets D and D' that differ at one user, say u. We denote X-SIM (i, j) in dataset D by $q(D, i, j)$ and the set of items in target domain, with quantified X-SIM values, by $I(i)$. Furthermore, we denote by $q(D, I(i))$ the set of X-SIM values between i and each item in $I(i)$. The global sensitivity (GS) is defined as $max_{D,D'}||q(D, i, j) - q(D', i, j)||_1$. Our PRS mechanism outputs an item j as a private replacement for i. We have the following:

$$\frac{Pr[PRS(i, I(i), q(D, I(i))) = j]}{Pr[PRS(i, I(i), q(D', I(i))) = j]} = \frac{exp(\frac{\epsilon \cdot q(D,i,j)}{2 \cdot GS})}{\sum\limits_{k \in I(i)} exp(\frac{\epsilon \cdot q(D,i,k)}{2 \cdot GS})} \div \frac{exp(\frac{\epsilon \cdot q(D',i,j)}{2 \cdot GS})}{\sum\limits_{k \in I(i)} exp(\frac{\epsilon \cdot q(D',i,k)}{2 \cdot GS})}$$

$$= \underbrace{\frac{exp(\frac{\epsilon \cdot q(D,i,j)}{2 \cdot GS})}{exp(\frac{\epsilon \cdot q(D',i,j)}{2 \cdot GS})}}_{\text{P}} \cdot \underbrace{\frac{\sum\limits_{k \in I(i)} exp(\frac{\epsilon \cdot q(D',i,k)}{2 \cdot GS})}{\sum\limits_{k \in I(i)} exp(\frac{\epsilon \cdot q(D,i,k)}{2 \cdot GS})}}_{\text{Q}}$$

$$P = exp(\frac{\epsilon \cdot (q(D, i, j) - q(D', i, j))}{2 \cdot GS}) \leq exp(\frac{\epsilon}{2})$$

$$Q = \frac{\sum\limits_{k \in I(i)} exp(\frac{\epsilon \cdot q(D',i,k)}{2 \cdot GS})}{\sum\limits_{k \in I(i)} exp(\frac{\epsilon \cdot q(D,i,k)}{2 \cdot GS})} \leq exp(\frac{\epsilon}{2})$$

Therefore, we get the following privacy guarantee:

$$\frac{Pr[PRS(i, I(i), q(D, I(i))) = j]}{Pr[PRS(i, I(i), q(D', I(i))) = j]} \leq exp(\epsilon)$$

Hence, PRS ensures ϵ-differential privacy (Definition 2). □

AlterEgo profile construction. In this step, we replace every item rated by Alice in $\mathcal{D}^S$ with the item in $\mathcal{D}^T$ returned by the PRS mechanism in the previous step. This item replacement scheme produces a private AlterEgo profile of Alice in the target domain.

Algorithm 8 *Private Replacement Selection Algorithm: PRS(i,I(i),X-Sim(I(i))) where I(i) is the set of items in the target domain with* X-SIM *values.*

```
Require: $\epsilon, i, I(i), X\text{-}Sim(I(i))$          ▷ $\epsilon$ : Privacy parameter
Global sensitivity for X-SIM:
GS = |X-SIM$_{max}$ − X-SIM$_{min}$| = 2
for item $j$ in $I(i)$ do
    Allocate probability as: $\frac{exp(\frac{\epsilon\cdot \text{X-SIM}(i,j)}{2\cdot GS})}{\sum_{k\in I(i)} exp(\frac{\epsilon\cdot \text{X-SIM}(i,k)}{2\cdot GS})}$
end for
Sample an element $t$ from $I(i)$ according to their probability.
return: $t$;          ▷ $\epsilon$-differentially private replacement for $i$
```

Note that this private AlterEgo profile protects the privacy of the straddlers, users who rated in both the domains, as the ratings of these users are used to compute the heterogeneous similarities leaving their privacy at risk [150]. In addition, if the application domains are typically owned by different companies like Netflix and Last.fm, then this mechanism aids the exchange of AlterEgo profiles while preventing curious or malicious users to infer the preferences of others.

D. Recommendation Phase

We now present the main steps of our recommendation scheme. Again, we first explain the non-private case followed by the private one.

NX-MAP recommendation. The AlterEgo profile of Alice is used along with the original profiles in the target domain to compute the top-k similar users for Alice and then compute recommendations for Alice leveraging the profiles of the k most similar users from the target domain as shown in Algorithm 1. The item-based version of X-MAP utilizes this AlterEgo profile and computes the recommendations as demonstrated in Algorithm 2.

Furthermore, the AlterEgo profile in the target domain also retains the temporal behavior [53] of the user in the source domain due to the item-to-item mapping. We incorporate this temporal behavior in the item-based version of X-MAP by adding a time-based weight to the ratings to improve the recommendation quality further. The predictions, weighted by the time-based parameter (α), for user u's ratings are computed as follows.[5]

$$Pred[i](t) = \bar{r}_i + \frac{\sum_{j \in N_k(i)\cap\mathcal{I}_u} \tau(i,j)\cdot(r_{u,j}-\bar{r}_j)\cdot e^{-\alpha(t-t_{u,j})}}{\sum_{j \in N_k(i)\cap\mathcal{I}_u} |\tau(i,j)|\cdot e^{-\alpha(t-t_{u,j})}} \tag{6.2}$$

Note that the prediction has a time-based relevance factor ($e^{-\alpha(t-t_{u,j})}$) with a decaying rate controlled by the parameter α to determine each rating's weight for the prediction computa-

---

[5] $N_k(i)$ denotes the top-k neighbors of item i.

tion. Here, $t_{u,j}$ denotes the *timestep*[6] when user A rated the item j. This specific time-based CF technique is applicable to the item-based CF approach as the prediction computation (Equation 6.2) for a user A is dependent only on her previous ratings for similar items and thereby leverages time as observed by A.

X-MAP recommendation. The private AlterEgo profile of Alice is used along with the original profiles in the target domain to compute the recommendations for Alice. To demonstrate the adaptability of our heterogeneous recommender, the recommendation step is integrated with a differentially private approach, inspired by [199, 200], to protect the privacy of users in the target domain against other curious users. We implemented both item-based and user-based versions of X-MAP. The item-based recommendation mechanism is demonstrated in Algorithm 10 which utilizes the PNSA mechanism (Algorithm 9). We first present our similarity-based sensitivity, required for the private approach, along with its correctness proof sketch. [7]

Definition 19 (LOCAL SENSITIVITY). *For any given function $f : \mathcal{R} \rightarrow \mathcal{R}$ and a dataset D, the Local Sensitivity of f is defined as $LS_f(D) = \max_{D'} \| f(D) - f(D') \|_1$, where D and D' are neighboring datasets which differ at one user profile.*

We define a rating vector $r_i = [r_{ai}, ..., r_{xi}, r_{yi}]$ which consists of all the ratings for an item $i \in D$. Similarly, we define a rating vector r'_i for $i \in D'$. Since we use adjusted-cosine for X-SIM, a rating r_{xi} is the result after subtracting the average rating of user x ($\bar{r}_x$) from the actual rating provided by x for an item i. The similarity-based sensitivity is formulated as follows.

Theorem 5 (SIMILARITY-BASED SENSITIVITY). *Given any score function $q : \mathcal{R} \rightarrow R$ and a dataset D, we formulate the similarity-based sensitivity corresponding to a score function $q_i(I, j)$ for a pair of items i and j as:*

$$SS(i,j) = max\{max_{u_x \in \mathcal{U}_{ij}}(\frac{r_{xi} \times r_{xj}}{\| r'_i \| \times \| r'_j \|}), max_{u_x \in \mathcal{U}_{ij}}(\frac{r_i \cdot r_j}{\| r'_i \| \times \| r'_j \|} - \frac{r_i \cdot r_j}{\| r_i \| \times \| r_j \|})\}$$

Proof. (The full proof is provided in Appendix §8.2.1 for interested readers.) The similarity-based sensitivity is measured by the maximal change in the similarity between two items when deleting a user's rating. The score function (q) for a pair of items i and j is defined as their similarity value ($s(i, j)$). First, SS is defined as:

$$SS(i,j) = max \| s(i,j) - s'(i,j) \|_1$$

---

[6] The timestep is a logical time corresponding to the actual timestamp of an event.

[7] Our similarity-based sensitivity is slightly different from the recommendation-aware one presented in [199, 200].

We arrive at the following equality after inserting the similarity values for $s(i,j)$.

$$s(i,j) - s'(i,j) = \frac{r_i \cdot r_j}{\parallel r_i \parallel \times \parallel r_j \parallel} - \frac{r'_i \cdot r'_j}{\parallel r'_i \parallel \times \parallel r'_j \parallel} = \frac{r_i \cdot r_j \times \parallel r'_i \parallel \times \parallel r'_j \parallel - r'_i \cdot r'_j \times \parallel r_i \parallel \times \parallel r_j \parallel}{\parallel r_i \parallel \times \parallel r_j \parallel \times \parallel r'_i \parallel \times \parallel r'_j \parallel} = \frac{P}{Q}$$

We assume that the profile of a user x, in D, is not present in D'. This user rated both i and j. Note that if this user rated one of these items or none, then the similarity value does not depend on the presence or absence of this user in the dataset. Hence, we get the inequality: $\parallel r'_i \parallel \times \parallel r'_j \parallel \leq \parallel r_i \parallel \times \parallel r_j \parallel$.

Recall that P= $(r_i \cdot r_j \times \parallel r'_i \parallel \times \parallel r'_j \parallel - r'_i \cdot r'_j \times \parallel r_i \parallel \times \parallel r_j \parallel)$ and Q=$(\parallel r_i \parallel \times \parallel r_j \parallel \times \parallel r'_i \parallel \times \parallel r'_j \parallel)$. Since $Q \geq 0$, we have two conditions depending on whether $P \geq 0$ or $P \leq 0$.

If $P \geq 0$, then

$$\parallel s(i,j) - s'(i,j) \parallel_1 \leq \frac{(r_i \cdot r_j - r'_i \cdot r'_j)}{\parallel r'_i \parallel \times \parallel r'_j \parallel} = \frac{r_{xi} \times r_{xj}}{\parallel r'_i \parallel \times \parallel r'_j \parallel}.$$

If $P \leq 0$, then

$$\parallel s(i,j) - s'(i,j) \parallel_1 \leq \frac{r_i \cdot r_j}{\parallel r'_i \parallel \times \parallel r'_j \parallel} - \frac{r_i \cdot r_j}{\parallel r_i \parallel \times \parallel r_j \parallel}.$$

Hence, we have the similarity-based sensitivity as:

$$SS(i,j) = max\{max_{u_x \in \mathcal{U}_{ij}}(\frac{r_{xi} \times r_{xj}}{\parallel r'_i \parallel \times \parallel r'_j \parallel}), max_{u_x \in \mathcal{U}_{ij}}(\frac{r_i \cdot r_j}{\parallel r'_i \parallel \times \parallel r'_j \parallel} - \frac{r_i \cdot r_j}{\parallel r_i \parallel \times \parallel r_j \parallel})\}$$

□

We use the notion of *truncated similarity* [199, 200] (Step 7 in Algorithm 9) along with our similarity-based sensitivity to enhance the quality of selected neighbors. The two theorems which prove that this truncated similarity along with our similarity-based sensitivity can enhance the quality of neighbors are as follows. (The detailed proofs for the following two theorems are available in the technical report hosted on our GitHub repository [186].)

Theorem 6. *Given any item i, we denote its k neighbors by $N_k(i)$, the maximal length of all the rating vector pairs by $|v|$, the minimal similarity among the items in $N_k(i)$ by $Sim_k(i)$ and the maximal similarity-based sensitivity between i and other items by SS. Then, for a small constant* $0 < \rho < 1$, *the similarity of all the items in $N_k(i)$ are larger than $(Sim_k(i) - w)$ with a probability at least* $1 - \rho$, *where $w = min(Sim_k(i), \frac{4k \times SS}{\epsilon'} \times ln\frac{k \times (|v| - k)}{\rho})$.*

Intuitively, Theorem 6 implies that the selected neighbors have similarities greater than some

Algorithm 9 *Private Neighbor Selection : PNSA(i,I,*Sim*(i)) where I is the set of all items.*

```
Require: $\epsilon', w, i, I, Sim(i)$, k                                ▷ $\epsilon'$ : Privacy
$C_1 = [j | s(i,j) \geq Sim_k(i) - w, j \in I]$
$C_0 = [j | s(i,j) < Sim_k(i) - w, j \in I]$
$w = min(Sim_k(i), \frac{4k \times SS}{\epsilon'} \times ln\frac{k \times (|v|-k)}{\rho})$
for N=1:$k$ do
    for item $j$ in $I$ do
        $SS(i,j) = max\{max_{u_x \in \mathcal{U}_{ij}}(\frac{r_{xi} \times r_{xj}}{\| r'_i \| \times \| r'_j \|}), max_{u_x \in \mathcal{U}_{ij}}(\frac{r_i \cdot r_j}{\| r'_i \| \times \| r'_j \|} - \frac{r_i \cdot r_j}{\| r_i \| \times \| r_j \|)})\}$
        $\widehat{Sim}(i,j) = max(Sim(i,j), Sim_k(i) - w)$
        Allocate probability as:                                       ▷ $\frac{\epsilon'}{2k}$-Privacy
            $\frac{exp(\frac{\epsilon' \cdot \widehat{Sim}(i,j)}{2k \times 2SS(i,j)})}{\sum_{l \in C_1} exp(\frac{\epsilon' \cdot \widehat{Sim}(i,l)}{2k \times 2SS(i,l)}) + \sum_{l \in C_0} exp(\frac{\epsilon' \cdot \widehat{Sim}(i,l)}{2k \times 2SS(i,l)})}$
    end for
    Sample an element $t$ from $C_1$ and $C_0$ without replacement according to their probability.
    $N_k(i) = N_k(i) \cup t$
end for
return: $N_k(i)$;
```

threshold value $(Sim_k(i) - w)$ with a high probability $(1 - \rho)$.

Theorem 7. *Given any item i, for a small constant $0 < \rho < 1$, all items with similarities greater than $(Sim_k(i) + w)$ are present in $N_k(i)$ with a probability at least $1 - \rho$ where $w = min(Sim_k(i), \frac{4k \times SS}{\epsilon'} \times ln\frac{k \times (|v|-k)}{\rho})$.*

Intuitively, Theorem 7 implies that the items with similarities greater than some threshold value $(Sim_k(i) + w)$ are selected as neighbors with a high probability $(1 - \rho)$.

Both theorems prove that the truncated similarity along with our similarity-based sensitivity provides neighbors of good quality while providing $\epsilon'/2$-differential privacy. The predictions are computed leveraging the PNCF mechanism (Algorithm 10) which adds Laplacian noise to provide $\epsilon'/2$-differential privacy. By the composition property of differential privacy, PNSA and PNCF together provide ϵ'-differential privacy. The item-based version of X-MAP includes the additional feature of *temporally relevant* predictions to boost the recommendation quality traded for privacy.

We provide here two illustrations (temporal dynamics and differential privacy) of the adaptability of our heterogeneous recommender. Since the AlterEgo profile could be considered as an actual profile in the target domain, thereby any homogeneous recommendation algorithm [2] like Matrix Factorization techniques, can be applied in the target domain to generate the recommendations. We provide a demonstration regarding how to use Spark-MLLIB's matrix factorization technique with X-MAP in our GitHub repository [186].

Algorithm 10 *Private Recommendation: PNCF(P_A, I) where P_A denotes the AlterEgo profile of Alice, and I denotes the set of all items.*

```
var P;                                   ▷ Dictionary with predictions for Alice
var $\tau$;                              ▷ User similarities
var $\bar{r}$;                           ▷ Average rating for each items
var $\epsilon'$                          ▷ Degree of privacy
var $N_k$                                ▷ Private neighbors using PNSA
for $i$ : item in $P_A$ do
    $N_k(i) = PNSA(i, I, Sim(i))$
    for $j$ : item in $N_k(i)$ do
        $P[j] = \bar{r}_j + \frac{\sum_{k\in N_k(j)}(\tau(k,j)+Lap(\frac{SS(k,j)}{\epsilon'/2}))\cdot(r_{A,k}-\bar{r}_k)}{\sum_{k\in N_k(j)}|\tau(k,j)+Lap(\frac{SS(k,j)}{\epsilon'/2})|}$
    end for
end for
$R_A$ = P.sortByValue(ascending=false);
return: $R_A[:N]$;                       ▷ Top-N recommendations for Alice
```

6.1.5 Implementation

We now describe our implementation of X-MAP. Figure 6.4 outlines the four main components of our implementation: *baseliner*, *extender*, *generator* and *recommender*. We describe each of these components along with their functionality.

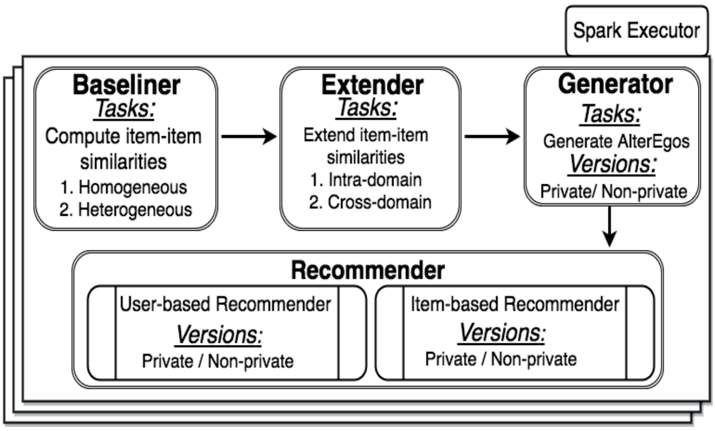


Figure 6.4 – *The components of* X-MAP*:* Baseliner, Extender, Generator, Recommender.

A. Baseliner

This component computes the baseline similarities leveraging the adjusted cosine similarity (Equation 6.1) between the items in the two domains. The baseliner splits the item-pairs based on whether both items belong to the same domain or not. If both items are from the same domain, then the item-pair similarities will be delivered as *homogeneous similarities*. If one of the items belongs to a different domain, then the item-pair similarities will be delivered as *heterogeneous similarities*. The baseline heterogeneous similarities are computed based on the user overlap. [8]

B. Extender

This component extends the baseline similarities both within a domain and across domains. The items in each domain are divided into three layers based on our layer-based pruning technique as shown in Figure 6.2. For every item in a specific layer, the extender computes the top-k similar items for the neighboring layers. For instance, for the items in the BB-layer of $\mathcal{D}^S$, the extender computes the top-k similar ones from items in the BB-layer in $\mathcal{D}^T$ and also the top-k similar ones from the items in the NB-layer in $\mathcal{D}^S$.

Intra-domain extension. In this step, the extender computes the X-SIM similarities for the items in the NN-layer in $\mathcal{D}^S$ and the items in the BB-layer of $\mathcal{D}^S$ via the NB-layer items of $\mathcal{D}^S$. Similar computations are performed for domain $\mathcal{D}^T$.

Cross-domain extension. After the previous step, the extender updates the NB and NN layers in both domains based on the new connections (top-k). Then, it updates the connections between the items in NB and BB layers in one domain and the items in NB and BB layers in the other one.

At the end of the execution, the extender outputs, for every item i in $\mathcal{D}^S$, a set of items $I(i)$ in $\mathcal{D}^T$ with some quantified (positive or negative) X-SIM values with i.

C. Generator

The generator performs the following computational steps.

Item mapping. The Generator maps every item in one domain (say $\mathcal{D}^S$) to its most similar item (for NX-MAP) or its private replacement (for X-MAP) in the other domain ($\mathcal{D}^T$). After, the completion of this step, every item in one domain has a replacement item in the other domain. [9]

Mapped user profiles. The Generator here creates an artificial profile (AlterEgo) of a user in a target domain $\mathcal{D}^T$ from her actual profile in the source domain $\mathcal{D}^S$ by replacing each item in her profile in $\mathcal{D}^S$ with its replacement in $\mathcal{D}^T$ as shown in Figure 6.3. Finally, after this step, the Generator outputs the AlterEgo profile of a user in the target domain where she might have little or no activity yet.

---

[8] These are the baseline similarities without any extension or enhancements.

[9] We could also choose a set of replacements for any item, using X-SIM, in the target domain to have more diversity.

D. Recommender

This component utilizes the artificial AlterEgo profile created by the Generator to perform the recommendation computation. It can implement any general recommendation algorithm for its underlying recommendation computation. In this work, we implemented user-based and item-based CF schemes. For NX-MAP, the recommender uses Algorithm 1 (user-based CF) or Algorithm 2 (item-based CF) in the target domain. For X-MAP, the recommender also uses the PNSA algorithm along with the PNCF algorithm to generate recommendations either in a user-based manner or in an item-based manner. Additionally, for both NX-MAP and X-MAP, the item-based CF recommender leverages the temporal relevance to boost the recommendation quality. It is important to note that X-MAP runs periodically in an offline manner to update the predicted ratings. The top-10 items (sorted by the predicted ratings), not-yet-seen by the current user, would be recommended to users in X-MAP.

6.1.6 Evaluation

We report here on our empirical evaluation of X-MAP on a cluster computing framework, namely Spark [189], with real world traces from Amazon [131] to analyze its prediction accuracy, privacy and scalability. We choose Spark as our cluster computing framework since the underlying data processing framework to support X-MAP must be scalable and fault-tolerant.

A. Experimental Setup

Experimental platform. We perform all the experiments on a cluster of 20 machines. Each machine is an Intel Xeon CPU E5520 @2.26GHz with 32 GB memory. The machines are connected through a 2xGigabit Ethernet (Broadcom NetXtremeII BCM5716).

Datasets. We now provide an overview of the datasets used in our experiments.

Amazon. We use two sets of real traces from Amazon datasets [131]: *movies* and *books*. We use the ratings for the period from 2011 till 2013. The movies dataset consists of 1,671,662 ratings from 473,764 users for 128,402 movies whereas the books dataset consists of 2,708,839 ratings from 725,846 users for 403,234 books. The ratings vary from 1 to 5 with an increment of 1. The overlapping users in these two datasets are those Amazon users who are present in both datasets and are ascertained using their Amazon customer-ids. The number of such overlapping users from both the domains is 78,201.

Movielens. We use the Movielens dataset (ML-20M) for evaluating performance of X-MAP within a single domain. This dataset consists of 20,000,263 ratings from 138,493 users for 27,278 movies. In this dataset, the ratings also vary from 1 to 5 with an increment of 1.

Evaluation metrics. We evaluate X-MAP along three complementary metrics: (1) the recommendation *quality* as perceived by the users in terms of prediction *accuracy*, (2) the degree of *privacy* provided to the end-users in terms of the privacy parameters (ϵ, ϵ'), and (3) the *scalability* in terms of *speedup* achieved in X-MAP when increasing the number of machines in the cluster.

Accuracy. We evaluate the accuracy of the predictions in terms of Mean Absolute Error (MAE). MAE computes the average absolute deviation between a predicted rating and the user's true rating. MAE is a standard metric used to evaluate state-of-the-art recommenders [84, 165]. We assume that the predicted rating for an item i is denoted by p_i and the actual rating is denoted by r_i in the test dataset. Then, the MAE for a test dataset, with $\mathcal{N}$ ratings, is computed as: $\frac{\sum_{i=1}^{\mathcal{N}} |p_i - r_i|}{\mathcal{N}}$. Given that r_{min} and r_{max} denotes the minimum and maximum ratings respectively, the following inequality always holds: $0 < MAE < (r_{max} - r_{min})$. The lower the MAE, the more accurate the predictions.

Privacy. Our differential privacy guarantees are parametrized as follows: ϵ for the PRS technique (Algorithm 8) used for AlterEgo generation and ϵ' for the PNCF (Algorithm 10) used for the private recommendation generation in X-MAP. According to the privacy literature [56, 199, 200], $\epsilon = 1$ or less would be suitable for privacy preserving purposes.

Speedup. We evaluate the speedup in terms of the time required for sequential execution (T_1) and the time required for parallel execution with p processors (T_p). Amdahl's law models the performance of speedup (S_p) as follows.

$$S_p = \frac{T_1}{T_p}$$

Due to the considerable amount of computations for heterogeneous recommendation on the Amazon dataset, we compare the speedup on p processors with respect to a minimum of 5 processors (T_5) instead of a sequential execution (T_1).

Competitors. We now present the recommenders against which we compare X-MAP. Existing recommendation schemes over several domains can be classified as follows.

Linked-domain personalization. The goal here is to recommend items in the target domain $(\mathcal{D}^T)$ by exploring rating preferences aggregated from both source and target domains, i.e, to recommend items in $\mathcal{I}^T$ to users in $\mathcal{U}^S$ based on the preferences of users in $\mathcal{U}^S \cup \mathcal{U}^T$ for items in $\mathcal{I}^S \cup \mathcal{I}^T$. In this approach, ratings from multiple domains are aggregated into a single domain. Then, a traditional CF mechanism can be applied over this aggregated single domain [157, 44]. ITEM-BASED-KNN is a linked-domain personalization approach [157, 44] where we use item-based collaborative filtering over the aggregated ratings from both the domains.

Heterogeneous recommendation. The goal here is to recommend items in $\mathcal{I}^T$ to users in $\mathcal{U}^S$ based on the preferences of $\mathcal{U}^S$ for $\mathcal{I}^S$, $\mathcal{U}^T$ for $\mathcal{I}^T$ and $\mathcal{U}^S \cap \mathcal{U}^T$ for $\mathcal{I}^S \cup \mathcal{I}^T$. In this approach, the user similarities are first computed in both source and target domains. These domain-related similarities are then aggregated into the overall heterogeneous similarities. Finally, the

k-nearest neighbors, used for recommendation computations, are selected based on these heterogeneous similarities [20]. In the REMOTEUSER approach [20], the user similarities in source domain are used to compute the k nearest neighbors for users who have not rated in the target domain. Finally, user-based collaborative filtering is performed.

Baseline prediction. For a sparse dataset, the baseline is provided by item-based average ratings [14] or user-based average ratings [116]. The goal here is to predict based on the average ratings provided by users in $\mathcal{U}^S \cup \mathcal{U}^T$ for items in $\mathcal{I}^S \cup \mathcal{I}^T$. One of the most basic prediction schemes is the ITEMAVERAGE scheme where we predict that each item will be rated as the average over all users who rated that item [14]. Note that though this technique gives a very good estimate of the actual rating but it is not personalized due to same predictions for all the users.

We compare X-MAP with these three other systems namely: ITEM-BASED-KNN, REMOTEUSER and ITEMAVERAGE.

Evaluation scheme. We partition the set of common users who rated both movies and books into *training* and *test* sets. For the test users, we hide their profile in the target domain (say books) and use their profile in the source domain (movies) to predict books for them. This strategy evaluates the accuracy of the predictions if the user did not rate any item in the target domain. Hence, we can evaluate the performance of X-MAP in the scenario where the test users did not rate any item in the target domain (cold-start). Additionally, if we hide part of the user profile in the target domain, then we can evaluate how X-MAP handles the scenario where the test users rated very few items in the target domain (sparsity). Furthermore, we denote the item-based variant of X-MAP as X-MAP-IB and the user-based variant as X-MAP-UB. Similarly for NX-MAP, we denote the item-based variant of NX-MAP as NX-MAP-IB and the user-based variant as NX-MAP-UB.

B. Temporal Dynamics

We observe the temporal effect of users, retained by the AlterEgos across domains, in X-MAP. We leverage the item-based recommender, and tune the temporal parameter α accordingly. Figure 6.5 demonstrates this temporal relevance effect where α varies between 0 (no temporal effect) to 0.2. Note that an item-based CF approach computes the predictions leveraging the target user's very few observed ratings on the nearest neighbors and given the very limited size of this set of ratings, any further amplification of α impacts the predictions negatively as it reduces the contribution of old ratings furthermore. We provide the optimally tuned parameter (α_o) for our experiments, shown in Figure 6.5, to achieve optimal recommendation quality.

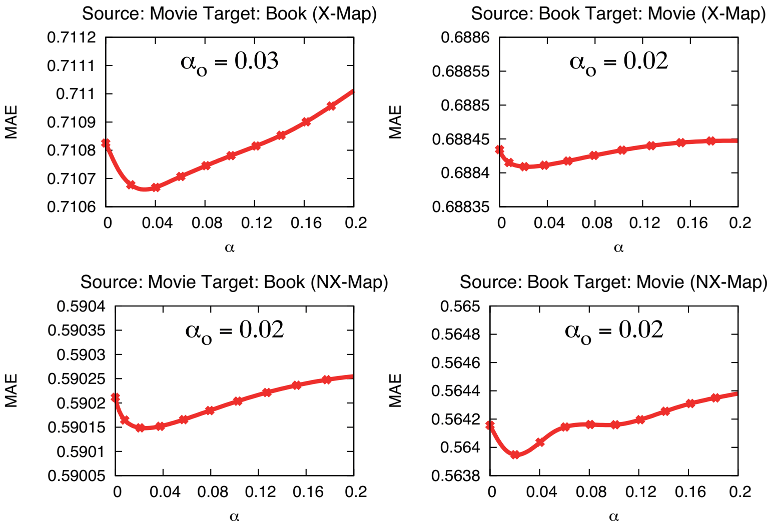


Figure 6.5 – *Temporal relevance (*X-Map, NX-Map*).*

C. Privacy

We tune the privacy parameters (ϵ, ϵ') for X-Map. Figures 6.6 and 6.7 demonstrate the effect of tuning the privacy parameters on the prediction quality in terms of MAE. We observe that the recommendation quality improves (lower MAE) as we decrease the degree of privacy (higher ϵ, ϵ'). It is important to note that X-Map inherently transforms to NX-Map as the privacy parameters increase furthermore (lower privacy guarantees). For the following experiments, we select the privacy parameters as follows. For X-Map-IB, we select $\epsilon = 0.3$ and $\epsilon' = 0.8$. For X-Map-UB, we select $\epsilon = 0.6$ and $\epsilon' = 0.3$. [10]

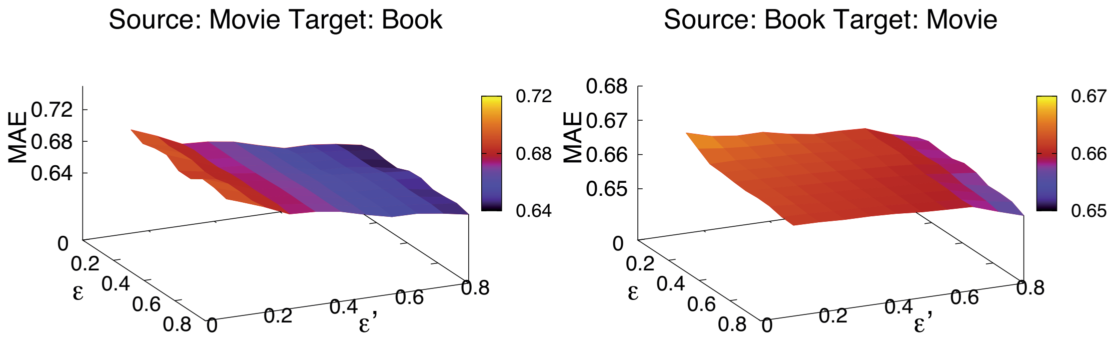


Figure 6.6 – *Privacy-quality trade-off in* X-Map-IB.

D. Accuracy

We now compare the accuracy of the predictions of X-Map and NX-Map with the competitors.

---

[10] These parameters are selected from a range of possible values providing quality close to the optimal one as observed from Figures 6.6 and 6.7.

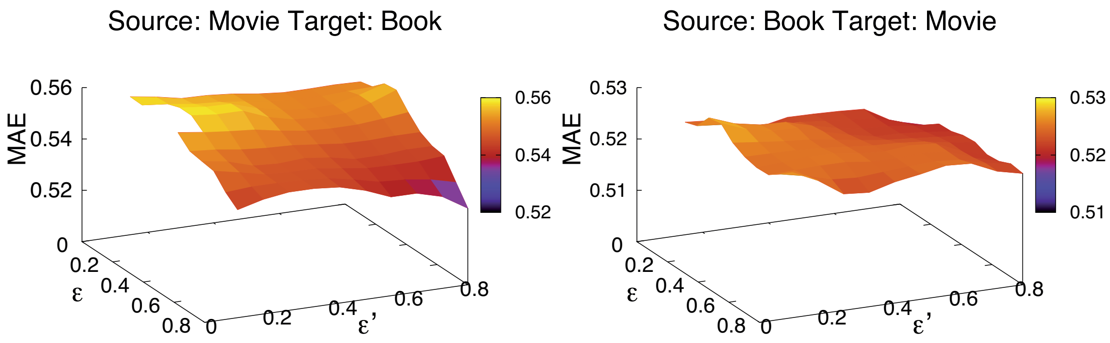


Figure 6.7 – *Privacy-quality trade-off in* X-Map-ub.

Impact of top-k neighbors. First, we evaluate the quality in terms of MAE when the size of k (neighbors in Equation 6.2) is varied. Figure 6.8(a) demonstrates that X-Map-ub and NX-Map-ub outperform the competitors by a significant margin of 30% where the source domain is book and the target domain is movie. Also, Figure 6.8(b) shows that X-Map performs better than the non-private competitors whereas NX-Map again outperforms the competitors by a margin of 18% where the source domain is movie and the target domain is book. A higher number of neighbors induces more connections across the domains (Figure 6.2) and hence enables X-Map to explore better meta-paths between items. Moreover, better meta-paths lead to better meta-path based similarities and thereby superior recommendation quality. We consider k as 50 for all further experiments.

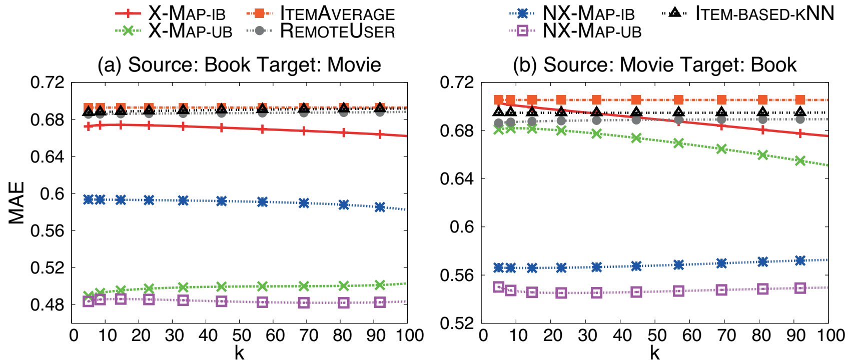


Figure 6.8 – *MAE comparison with varying* k.

Impact of overlap. We now evaluate how X-Map and NX-Map perform when the number of users in the overlap increases. Intuitively, a good approach should provide better accuracy as more and more users connect the domains. These increasing connections improve the baseline heterogeneous similarities which are then leveraged by X-Sim to generate better meta-path based similarities across the domains. Figure 6.9 shows that the prediction error of X-Map decreases as there are more users connecting the domains. This observation demonstrates that the quality of the AlterEgo profiles improves when the overlap size increases. Furthermore, we observe in Figure 6.9(a) that the user-based models show more improvement than the item-based ones. This behavior occurs as the item similarities are more static than the user similarities [94].

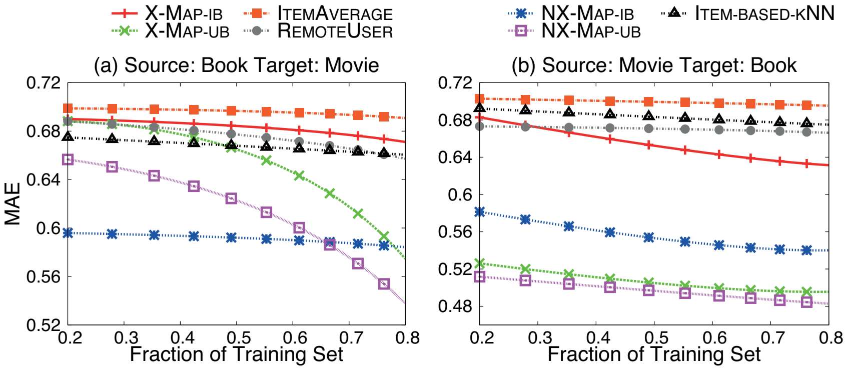


Figure 6.9 – *MAE comparison (Overlap size).*[11]

Impact of sparsity. We now evaluate how X-MAP performs when the size of the training profile of a user, in the target domain, increases from a minimum of 0 (cold-start situation) to a maximum of 6 (low sparsity), in addition to her profile in the source domain [12]. This experiment also highlights the performance of X-MAP when the sparsity of the dataset decreases. Additionally, we evaluate the accuracy improvement of X-MAP over a single domain solution, *item-based* KNN *in the target domain* denoted by KNN-SD, as well as over a heterogeneous solution, *item-based* KNN *in the aggregated domain* denoted by KNN-CD. Figure 6.10 demonstrates that KNN-SD and KNN-CD are outperformed by NX-MAP and X-MAP. Furthermore, we observe a relatively fast improvement for our non-private item-based technique (NX-MAP-IB) due to the improvement in item similarities with lower sparsity.

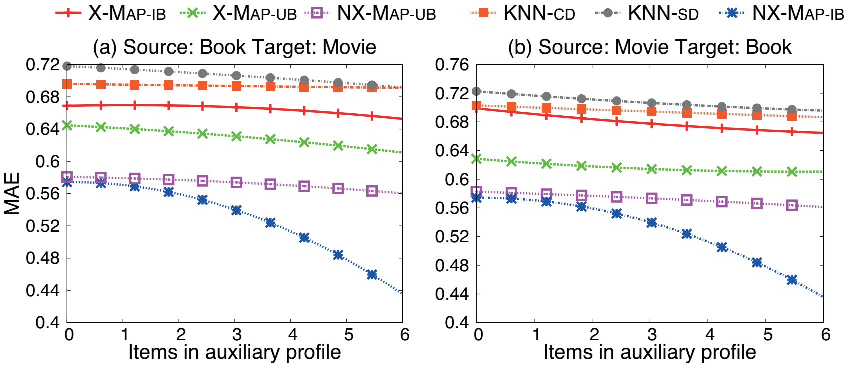


Figure 6.10 – *MAE comparison based on profile size.*

---

[11] Training set size denotes overlap size.

[12] We consider only those users who rated at least 10 products in each domain.

E. Homogeneity

We now evaluate the ability of X-MAP to be applied to a homogeneous setting consisting of a single domain. Depending on the structural property of the data (e.g., genres), any domain could be partitioned into multiple sub-domains. For this experiment, we use the ML-20M dataset which consists of 19 different genres. We partition this dataset into two sub-domains D_1 and D_2 by sorting the genres based on the movie counts per genre and allocating alternate sorted genres to the sub-domains as shown in Table 6.1. Note that a movie can have multiple genres. If a movie m belongs to both the sub-domains, we add it to the sub-domain which has the most number of genres overlapping with m's set of genres and to any of the two sub-domains in case of equal overlap with both sub-domains. Sub-domain D_1 consists of $15,119$ movies with $138,492$ users whereas sub-domain D_2 consists of $11,383$ movies with $138,483$ users.

D_1		D_2	
Genres	**Movie counts**	**Genres**	**Movie counts**
Drama	13344	Comedy	8374
Thriller	4178	Romance	4127
Action	3520	Crime	2939
Horror	2611	Documentary	2471
Adventure	2329	Sci-Fi	1743
Mystery	1514	Fantasy	1412
War	1194	Children	1139
Musical	1036	Animation	1027
Western	676	Film-Noir	330
Other	196	–	–

Table 6.1 – *Sub-domains (D_1 and D_2) based on genres in Movielens 20M dataset.*

We compare X-MAP and NX-MAP with Alternating Least Square from MLLIB (MLLIB-ALS). We observe from Table 6.2 that NX-MAP significantly outperforms MLLIB-ALS whereas X-MAP, even with the additional privacy overhead, almost retains the quality of non-private MLLIB-ALS.

	NX-MAP	X-MAP	MLLIB-ALS
MAE	**0.6027**	0.6830	0.6729

Table 6.2 – *MAE comparison (homogeneous setting on ML-20M dataset).*

F. Scalability

We evaluate the scalability of X-MAP in terms of the speedup achieved with an increasing number of computational nodes. We also compare our scalability with a state-of-the-art homogeneous recommender leveraging Spark to implement *Alternating-Least-Squares* based matrix factorization (MLLIB-ALS). For the ALS recommender, we use the aggregated ratings over both the domains (linked-domain personalization). Figure 6.11 demonstrates the near-linear speedup of X-MAP. Additionally, we see that X-MAP outperforms the scalability achieved by MLLIB-ALS. Note that X-MAP is periodically executed offline and the computation time for the recommendations, corresponding to all the users in the test set, is around 810 seconds on 20 nodes.

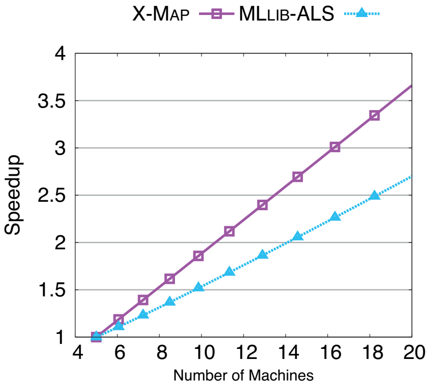


Figure 6.11 – *Scalability of* X-MAP.

G. Online Deployment

We deployed an online recommendation platform (http://x-map.work/) leveraging X-SIM and made it available to users. We observe that this recommender indeed provides book recommendations like *Shutter Island: A Novel* when the user queries for the movie *Inception*. Besides, it also recommends the *Shutter Island* movie as a homogeneous recommendation. We observe similar results for multiple other queries.

We deployed a real-time recommender implementing the underlying X-SIM and made it available to internet users. We collected user feedback for a duration of one week which is summarized in Figure 6.12. The x-axis denotes the score, provided by the user, in terms of a rating scale (1-5) with increment of 0.5 and the y-axis denotes the percentage of the total number of users. This preliminary study shows that the user satisfaction level is high.

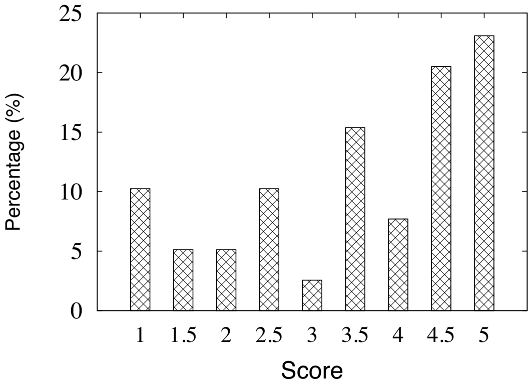


Figure 6.12 – *Feedback from* 51 *users over 1 week.*

6.1.7 Related Work

Heterogeneous trends. Research on heterogeneous recommendation is relatively new. There are, however, a few approaches to tackle the problem which we discuss below.

Smart User Models. González et al. introduced the notion of Smart User Models (SUMs) [69]. The idea is to aggregate heterogeneous information to build user profiles that are applicable across different domains. SUMs rely on users' emotional context which are, however, difficult to capture. Additionally, it has been shown that users' ratings vary frequently with time depending on their emotions [8].

Web Monitoring. Hyung et al. designed a web agent which profiles user preferences across multiple domains and leverages this information for personalized web support [108]. Tuffield et al. proposed Semantic Logger, a meta-data acquisition web agent that collects and stores any information (from emails, URLs, tags) accessed by the users [179]. However, web agents are considered a threat to users' privacy as users' data over different e-commerce applications are stored in a central database administered by the web agent.

Cross-domain Mediation. Berkovsky et al. [20] proposed the idea of cross-domain mediation to compute recommendations by aggregating data from several recommenders. We showed empirically that X-MAP outperforms cross-domain mediation in Figures 6.8 and 6.9.

In contrast, X-MAP introduces a new trend in heterogeneous personalization in the sense that the user profile from a source domain is leveraged to generate an *artificial* AlterEgo profile in a target domain. The AlterEgo profiles can even be exchanged between e-commerce companies like Netflix, Last.fm thanks to the privacy guarantee in X-MAP.

Merging preferences. One could also view the heterogeneous recommendation problem as that of merging single-domain user preferences. Through this viewpoint, several approaches can be considered which we discuss below.

Rating aggregation. This approach is based on aggregating user ratings over several domains into a single multi-domain rating matrix [21, 20]. Berkovsky et al. showed that this approach can tackle cold-start problems in collaborative filtering [21]. We showed empirically that X-MAP easily outperforms such rating aggregation based approaches [20].

Common representation. This approach is based on a common representation of user preferences from multiple domains either in the form of a *social tag* [177] or *semantic relationships between domains* [124]. Shi et al. developed a Tag-induced Cross-Domain Collaborative Filtering (TAGCDCF) to overcome cold-start problems in collaborative filtering [167]. TAGCDCF exploits shared tags to link different domains. They thus need additional tags to bridge the domains. X-MAP can bridge the domains based on the ratings provided by users using its novel X-SIM measure without requiring any such additional information which is difficult to collect in practice.

Linked preferences. This approach is based on linking users' preferences in several domains [44]. We showed empirically that X-MAP outperforms such linked preference based approaches [44] in Figures 6.8 and 6.9.

Domain-independent features. This approach is based on mapping user preferences to domain-independent features like *personality types* [34] or *user-item interactions* [125]. This approach again requires additional information like *personality scores* which might not be available for all users.

6.1.8 Conclusion

We presented X-MAP, a scalable and private heterogeneous recommender. X-MAP leverages a novel similarity metric X-SIM, identifying similar items across domains based on meta-paths, to generate AlterEgo profiles of users in domains where these users might not have any activity yet. We demonstrated that X-MAP performs better in terms of recommendation quality than alternative heterogeneous recommenders [14, 20, 44]. (Although, not surprisingly, there is a trade-off between quality and privacy.)

6.2 Content-enabled Heterogeneous Recommendations

In the previous section, we introduced a heterogeneous recommender which employs only the user-item interactions. However, it is also possible to perform content-enabled heterogeneous recommendations when the content is available about the users (e.g., demography, time-varying preferences) or items (e.g., popularity, price). These features could be explored *concurrently* to enable heterogeneous recommendations.

In this promising direction, we explore the notion of TRACKERS which enables us to incorporate these factors concurrently. We also capture item-to-item relations, based on their consumption sequence, leveraging neural embeddings for offers in our OFFER2VEC algorithm (similar to DEEPCIP in §4.2). We then introduce BOOSTJET, a novel recommender which integrates the TRACKERS along with the neural embeddings using MATRIXNET [79], an efficient distributed implementation of gradient boosted decision tree, to improve the recommendation quality significantly.

More precisely, BOOSTJET computes the recommendations as follows. First, BOOSTJET generates the TRACKERS which are statistical aggregates of users' activity capturing factors of different types (content, temporal, demographic, or monetary). Second, BOOSTJET generates the offer embeddings to capture the higher-dimensional relation between different offers in a given shop based on their consumption order by different users. These embeddings are generated using the proposed OFFER2VEC algorithm, our modification of DOC2VEC [114], by treating each *user session*, in a given shop, as a *document* and offers in this session as *words*. Finally, with the help of MATRIXNET we combine these features by posing the recommendation task as a classification problem in BOOSTJET, i.e., the recommendation task is to compute the likelihood probabilities for any given user corresponding to unseen items in a given application and then provide the highly predicted ones as recommendations to the user.

We evaluate BOOSTJET on Yandex's dataset, collecting online behavior from 14 million online users over 1250 different e-commerce applications, to demonstrate the practicality of BOOSTJET in terms of recommendation quality as well as scalability. Further details about this work is available in [145] for interested readers.

PART V

Thesis Conclusions and Remarks

In this part of the thesis, we summarize the main contributions of this thesis and also provide some concluding remarks regarding its implications on personalization at a high level. We also discuss some interesting directions for future work that the contributions of this thesis enable.

7 Concluding Remarks

We conclude this dissertation by discussing the outcomes and implications of the various contributions presented in this thesis along with the potential extensions for future work.

7.1 Summary and Implications

We first recall that the primary challenges in designing personalization services are *scalability*, *privacy*, and *heterogeneity*. We address these challenges step-by-step in each part of this thesis.

In the first part of this dissertation (Chapters 3 and 4), we address the scalability challenge. First, we significantly reduce the number of computations by leveraging an *iterative biased sampling* technique in HYREC (§3.1). Furthermore, HYREC democratizes these biased samples, personalized for each user, to the devices of the users for updating the recommender. We also briefly explore the extension of this democratization technique to classical machine learning applications using HYML (§3.2). Second, we take an *incremental* approach where we incorporate the updates to the recommender system in an incremental manner employing only the new incoming events (e.g., ratings or consumption events). We present two approaches depending on the type of feedback (i.e., users' preferences) which could be either explicit (e.g., numerical or binary ratings) or implicit (e.g., time for the consumption events). I-SIM (§4.1) enables incremental updates for explicit feedback whereas CIP (§4.2) enables incremental updates for implicit feedback.

In the second part of this dissertation (Chapter 5), we tackle the privacy challenge. We consider two levels of privacy. The first one is *user-level privacy* which deals with protecting the privacy of users from other curious users whereas the second one is *system-level privacy* which deals with protecting the privacy of users from the service provider itself. Concerning the user-level privacy, we introduce the notion of distance-based differential privacy (D2P) in §5.1 which strengthens the notion of differential privacy for recommender systems. We also present a brief overview of X-REC in §5.2 which ensures the privacy of users against the service providers (*system-level privacy*) or other users (*user-level privacy*) while preserving recommendation quality.

In the third part of this dissertation (Chapter 6), we tackle the heterogeneity challenge. With this objective in mind, we introduce X-MAP which is a novel heterogeneous recommender system employing meta-path-based transitive closure of inter-item similarities across several domains to provide recommendations across multiple domains. Additionally, we show that X-MAP enables differentially private recommendations and also easily scales out on multiple machines. We also briefly explore the impact of content towards heterogeneous recommendations by employing statistical aggregates of content-based features of users (e.g., demography, temporal preferences) or items (e.g., popularity, price).

At a high level, this dissertation takes a step in personalizing the Web in the sense that AlterEgos of any web user could be now extracted across various Internet applications and then employed to identify a personalized slice of Internet for web navigation of that user. Moreover, such heterogeneous web personalization could be now provided to the users without risking their privacy thanks to the private AlterEgos (Chapter 6) and distance-based differential privacy (Chapter 5). The scalability of the different personalization schemes, presented in this dissertation, also ensure that the personalized web slices for users could be updated in real-time depending on their recent explicit or implicit preferences (which might vary significantly overtime).

7.2 Future Work

We now discuss some potential directions for future research that build on the work presented in this dissertation.

Extension to other ML applications. Most of the work presented in this thesis could be extended to various other ML applications. We provide one demonstration of such extension where we show how we can extend the democratization idea used in HYREC to enable ML on users' devices (Chapter 3). The notion of distance-based differential privacy (Chapter 5) could also be explored in the context *event-level privacy* [103] for ML applications where the distance could be defined based on the input features and the output labels. For e.g., it is possible to design a privacy-aware classifier such that it can distinguish between *bikes* and *cars* where the distance could be defined in such a way that the *bikes* class is a superset of different types of bikes (e.g., road bikes, mountain bikes, racing bikes) and the *cars* class is a superset of different types of cars (e.g., sports cars, family cars, luxury cars). Such a classifier would preserve privacy in the sense that images could also reveal various personal details e.g., location [123]. It would also be interesting to employ techniques for *system-level privacy* concerning various ML applications, e.g., ML over encrypted data [26, 72].

Private incremental updates. The privacy guarantees presented in this dissertation concern with static databases of user-item interaction events (Chapter 5). However, we also introduced the notion of incrementality to handle scalability (Chapter 4). If we apply the privacy-preserving techniques over the complete aggregated data during the incremental updates, then the computation overhead for privacy might significantly affect the total time for performing the incremental updates. This limitation calls for *incremental privacy-aware* techniques that would support such incremental updates without significant overhead. There has already been some recent work in this direction for ML to address the problems of *private incremental Expected Risk Minimization* (ERM) and *private incremental regression* [101]. Hence, it would be interesting to explore such incremental private solutions for ML to design private and incremental recommenders.

Energy-efficient recommenders. We briefly demonstrate the impact of our approach on reducing the energy consumption in §4.1. This impact is also intuitive due to the nature of incrementality incorporated in the computations to update the recommender. ML applications are also extremely resource-greedy which leads to significant energy consumption. Recently, there has been some work in designing various compression techniques like quantization [6] or knowledge distillation [86] to significantly reduce the ML workload and hence also achieves energy-efficiency. Similar techniques could be employed along with our incremental approaches for recommenders to improve the energy-efficiency furthermore.

PART VI

Appendices

In this part of the thesis, we provide some supplementary materials (e.g., detailed correctness proofs of algorithms, additional experiments) for interested readers.

8 Appendices

8.1 Appendix A: I-Sim

8.1.1 Correctness proofs

Theorem 1 (P_{ij} INCREMENTAL UPDATE)**.** *Let $\Delta\mathcal{U}_i^t$ denote the set of users who newly rated i at timestep t, i.e., $\Delta\mathcal{U}_i^t = \mathcal{U}_i^t \setminus \mathcal{U}_i^{t-1}$, then the time complexity for updating $P_{ij}(t)$ is $\mathcal{O}(|\Delta\mathcal{U}_i^t| + |\Delta\mathcal{U}_j^t|)$.*

Proof. We obtain a recursive relation between $P_{ij}(t)$ and $P_{ij}(t-1)$ by decomposing $P_{ij}(t)$ as follows.

$$\begin{aligned}
P_{ij}(t) &= \sum_{u \in \mathcal{U}_i^t \cap \mathcal{U}_j^t} f_{ui}^{\alpha}(t)(r_{ui} - \bar{r}_u(t)) f_{uj}^{\alpha}(t)(r_{uj} - \bar{r}_u(t)) \\
&= \sum_{u \in \Delta\mathcal{U}_i^t \cap \mathcal{U}_j^{t-1}} (r_{ui} - \bar{r}_u(t)) f_{uj}^{\alpha}(t)(r_{uj} - \bar{r}_u(t)) + \sum_{u \in \mathcal{U}_i^{t-1} \cap \Delta\mathcal{U}_j^t} f_{ui}^{\alpha}(t)(r_{ui} - \bar{r}_u(t))(r_{uj} - \bar{r}_u(t)) \\
&\quad + \sum_{u \in \Delta\mathcal{U}_i^t \cap \Delta\mathcal{U}_j^t} (r_{ui} - \bar{r}_u(t))(r_{uj} - \bar{r}_u(t)) + \sum_{u \in \mathcal{U}_i^{t-1} \cap \mathcal{U}_j^{t-1}} f_{ui}^{\alpha}(t)(r_{ui} - \bar{r}_u(t)) f_{uj}^{\alpha}(t)(r_{uj} - \bar{r}_u(t)) \\
&= \Delta P_{ij}(t) + e^{-2\alpha} P'_{ij}(t-1)
\end{aligned}$$

In the above mathematical expression, we have absorbed the first three summations into the term $\Delta P_{ij}(t)$ and defined the last term as $P'_{ij}(t-1)$. Furthermore, we have: $\epsilon(t) \triangleq \bar{r}_u(t) - \bar{r}_u(t-1)$. Note that $\epsilon(t) \equiv \epsilon_u(t)$ varies for each user and alters marginally over consecutive timesteps: $\epsilon(t) = \epsilon(t-1) + \Delta\epsilon$. We rewrite $P'_{ij}(t-1)$ as follows.

$$\begin{aligned}
P'_{ij}(t-1) &= \sum_{u \in \mathcal{U}_{ij}^{t-1}} f_{ui}^{\alpha}(t-1)(r_{ui} - \bar{r}_u(t)) f_{uj}^{\alpha}(t-1)(r_{uj} - \bar{r}_u(t)) \\
&= \sum_{u \in \mathcal{U}_{ij}^{t-1}} f_{ui}^{\alpha}(t-1)(r_{ui} - \bar{r}_u(t-1)) f_{uj}^{\alpha}(t-1)(r_{uj} - \bar{r}_u(t-1))
\end{aligned}$$

$$
\begin{aligned}
&- \sum_{u \in \mathcal{U}_{ij}^{t-1}} (\epsilon(t-1) + \Delta\epsilon) f_{ui}^{\alpha}(t-1) f_{uj}^{\alpha}(t-1)(r_{ui} - \bar{r}_u(t-1)) \\
&- \sum_{u \in \mathcal{U}_{ij}^{t-1}} (\epsilon(t-1) + \Delta\epsilon) f_{ui}^{\alpha}(t-1) f_{uj}^{\alpha}(t-1)(r_{uj} - \bar{r}_u(t-1)) \\
&+ \sum_{u \in \mathcal{U}_{ij}^{t-1}} (\epsilon(t-1) + \Delta\epsilon)^2 \cdot f_{ui}^{\alpha}(t-1) f_{uj}^{\alpha}(t-1)
\end{aligned}
$$

where $\mathcal{U}_{ij}^{t-1}$ denotes $\mathcal{U}_i^{t-1} \cap \mathcal{U}_j^{t-1}$.

In the following, we ignore negligibly small higher order terms with the multiplicative factor $\Delta\epsilon \cdot f_{ui}^{\alpha}(t) \cdot f_{uj}^{\alpha}(t)$ as each of the terms $\{\Delta\epsilon, f_{ui}^{\alpha}(t), f_{uj}^{\alpha}(t)\} << 1$.

$$
\begin{aligned}
P'_{ij}(t-1) &= \sum_{u \in \mathcal{U}_{ij}^{t-1}} f_{ui}^{\alpha}(t-1)(r_{ui} - \bar{r}_u(t)) f_{uj}^{\alpha}(t-1)(r_{uj} - \bar{r}_u(t)) \\
&= \sum_{u \in \mathcal{U}_{ij}^{t-1}} f_{ui}^{\alpha}(t-1)(r_{ui} - \bar{r}_u(t-1)) f_{uj}^{\alpha}(t-1)(r_{uj} - \bar{r}_u(t-1)) \\
&\quad - \sum_{u \in \mathcal{U}_{ij}^{t-1}} \epsilon(t-1) f_{ui}^{\alpha}(t-1) f_{uj}^{\alpha}(t-1)(r_{ui} - \bar{r}_u(t-1)) \\
&\quad - \sum_{u \in \mathcal{U}_{ij}^{t-1}} \epsilon(t-1) f_{ui}^{\alpha}(t-1) f_{uj}^{\alpha}(t-1)(r_{uj} - \bar{r}_u(t-1)) \\
&\quad + \sum_{u \in \mathcal{U}_{ij}^{t-1}} \{\epsilon(t-1)\}^2 \cdot f_{ui}^{\alpha}(t-1) f_{uj}^{\alpha}(t-1)
\end{aligned}
$$

We introduce two adjustment terms L, M in the following. Note that these adjustment terms incorporate the behavioral drift, captured by $\epsilon(t)$, in I-SIM.

$$
L_{ij}(t) = \sum_{u \in \mathcal{U}_{ij}^{t}} \epsilon(t) f_{ui}^{\alpha}(t) f_{uj}^{\alpha}(t)[(r_{ui} - \bar{r}_u(t)) + (r_{uj} - \bar{r}_u(t))] = \sum_{u \in \mathcal{U}_{ij}^{t}} \epsilon(t) f_{ui}^{\alpha}(t) f_{uj}^{\alpha}(t)(r_{ui} + r_{uj} - 2\bar{r}_u(t)) \tag{8.1}
$$

$$
L_i(t) = 2 \sum_{u \in \mathcal{U}_i^{t}} \epsilon(t) f_{ui}^{2\alpha}(t)(r_{ui} - \bar{r}_u(t))
$$

We introduce the other adjustment term M which is as follows.

$$
M_{ij}(t) = \sum_{u \in \mathcal{U}_{ij}^{t}} \epsilon(t)^2 \cdot f_{ui}^{\alpha}(t) f_{uj}^{\alpha}(t) \tag{8.2}
$$

$$
M_i(t) = \sum_{u \in \mathcal{U}_i^{t}} \epsilon(t)^2 \cdot f_{ui}^{2\alpha}(t) \tag{8.3}
$$

We can thus compute $P_{ij}(t)$ incrementally as follows.

$$
P_{ij}(t) = \Delta P_{ij}(t) + e^{-2\alpha}[P_{ij}(t-1) - L_{ij}(t-1) + M_{ij}(t-1)]
$$

We can have a similar incremental update relation for $L_{ij}(t)$ as follows.

$$\begin{aligned}
L_{ij}(t) &= \sum_{u\in\mathcal{U}_{ij}^{t}} \epsilon(t) f_{ui}^{\alpha}(t) f_{uj}^{\alpha}(t)(r_{ui}+r_{uj}-2\bar{r}_u(t)) \\
&= \Delta L_{ij}(t) + e^{-2\alpha} \sum_{u\in\mathcal{U}_{ij}^{t-1}} (\epsilon(t-1)+\Delta\epsilon) f_{ui}^{\alpha}(t-1) f_{uj}^{\alpha}(t-1)(r_{ui}+r_{uj}-2\bar{r}_u(t)) \\
&= \Delta L_{ij}(t) + e^{-2\alpha} \sum_{u\in\mathcal{U}_{ij}^{t-1}} (\epsilon(t-1)+\Delta\epsilon) f_{ui}^{\alpha}(t-1) f_{uj}^{\alpha}(t-1)(r_{ui}+r_{uj}-2\bar{r}_u(t-1)-2(\epsilon(t-1)+\Delta\epsilon)) \\
&= \Delta L_{ij}(t) + e^{-2\alpha} \sum_{u\in\mathcal{U}_{ij}^{t-1}} (\epsilon(t-1)+\Delta\epsilon) f_{ui}^{\alpha}(t-1) f_{uj}^{\alpha}(t-1)(r_{ui}+r_{uj}-2\bar{r}_u(t-1)) \\
&\quad - 2e^{-2\alpha} \sum_{u\in\mathcal{U}_{ij}^{t-1}} (\epsilon(t-1)+\Delta\epsilon)^2 f_{ui}^{\alpha}(t-1) f_{uj}^{\alpha}(t-1)
\end{aligned}$$

Again, we ignore negligibly small higher order terms with the multiplicative factor $\Delta\epsilon \cdot f_{ui}^{\alpha}(t) \cdot f_{uj}^{\alpha}(t)$ as each of the terms $\{\Delta\epsilon, f_{ui}^{\alpha}(t), f_{uj}^{\alpha}(t)\} << 1$, and thereby get the following:

$$\begin{aligned}
L_{ij}(t) &= \sum_{u\in\mathcal{U}_{ij}^{t}} \epsilon(t) f_{ui}^{\alpha}(t) f_{uj}^{\alpha}(t)(r_{ui}+r_{uj}-2\bar{r}_u(t)) \\
&= \Delta L_{ij}(t) + e^{-2\alpha} \sum_{u\in\mathcal{U}_{ij}^{t-1}} \epsilon(t-1) f_{ui}^{\alpha}(t-1) f_{uj}^{\alpha}(t-1)(r_{ui}+r_{uj}-2\bar{r}_u(t)) \\
&= \Delta L_{ij}(t) + e^{-2\alpha} \sum_{u\in\mathcal{U}_{ij}^{t-1}} \epsilon(t-1) f_{ui}^{\alpha}(t-1) f_{uj}^{\alpha}(t-1)(r_{ui}+r_{uj}-2\bar{r}_u(t-1)-2\epsilon(t-1)) \\
&= \Delta L_{ij}(t) + e^{-2\alpha} \sum_{u\in\mathcal{U}_{ij}^{t-1}} \epsilon(t-1) f_{ui}^{\alpha}(t-1) f_{uj}^{\alpha}(t-1)(r_{ui}+r_{uj}-2\bar{r}_u(t-1)) \\
&\quad - 2e^{-2\alpha} \sum_{u\in\mathcal{U}_{ij}^{t-1}} (\epsilon(t-1))^2 f_{ui}^{\alpha}(t-1) f_{uj}^{\alpha}(t-1)
\end{aligned}$$

We get the recursive relation for $L_{ij}(t)$ as follows.

$$L_{ij}(t) = \Delta L_{ij}(t) + e^{-2\alpha}[L_{ij}(t-1) - 2M_{ij}(t-1)]$$

where the $\Delta L_{ij}(t)$ is as follows.

$$\begin{aligned}
\Delta L_{ij}(t) = &\sum_{u\in\Delta\mathcal{U}_i^{t}\cap\mathcal{U}_j^{t-1}} \epsilon(t) f_{uj}^{\alpha}(t)(r_{ui}+r_{uj}-2\bar{r}_u(t)) + \sum_{u\in\mathcal{U}_i^{t-1}\cap\Delta\mathcal{U}_j^{t}} \epsilon(t) f_{ui}^{\alpha}(t)(r_{ui}+r_{uj}-2\bar{r}_u(t)) \\
&+ \sum_{u\in\Delta\mathcal{U}_i^{t}\cap\Delta\mathcal{U}_j^{t}} \epsilon(t)(r_{ui}+r_{uj}-2\bar{r}_u(t))
\end{aligned}$$

We can get a similar recursive relation for $M_{ij}(t)$ as follows.

$$M_{ij}(t) = \Delta M_{ij}(t) + e^{-2\alpha} M_{ij}(t-1)$$

where the $\Delta M_{ij}(t)$ is as follows.

$$\Delta M_{ij}(t) = \sum_{u \in \Delta\mathcal{U}_i^t \cap \mathcal{U}_j^{t-1}} \epsilon(t)^2 f_{uj}^{\alpha}(t) + \sum_{u \in \mathcal{U}_i^{t-1} \cap \Delta\mathcal{U}_j^t} \epsilon(t)^2 f_{ui}^{\alpha}(t) + \sum_{u \in \Delta\mathcal{U}_i^t \cap \Delta\mathcal{U}_j^t} \epsilon(t)^2$$

We observe that the terms to be incrementally updated in order to update $P_{ij}(t)$, namely $\Delta P_{ij}(t)$, $\Delta L_{ij}(t)$ and $\Delta M_{ij}(t)$, have a time complexity bounded by $\mathcal{O}(|\Delta\mathcal{U}_i^t| + |\Delta\mathcal{U}_j^t|)$. Note that if $P_{ij}(t)$ was updated non-incrementally then the time complexity would be $\mathcal{O}(|\mathcal{U}_i^t \cap \mathcal{U}_j^t|)$. With each timestep, the number of new ratings for i ($|\Delta\mathcal{U}_i^t|$) tends to be significantly smaller than the total number of ratings for i ($|\mathcal{U}_i^t|$). The difference is huge even for the average case as $|\mathcal{U}_i^t|$ can be of the order of all users in the system. ☐

We now provide the update relation for $Q_i(t)$.

Theorem 2 (Q_i INCREMENTAL UPDATE). *Given that $\Delta\mathcal{U}_i^t$ denotes the set of users who newly rated i at timestep t, i.e. $\Delta\mathcal{U}_i^t = \mathcal{U}_i^t \setminus \mathcal{U}_i^{t-1}$, then the time complexity for updating $Q_i(t)$ is $\mathcal{O}(|\Delta\mathcal{U}_i^t|)$.*

Proof. We again obtain a recursive relation between $Q_i(t)$ and $Q_i(t-1)$ by decomposing $Q_i(t)$ as follows.

$$\begin{aligned}
Q_i(t) &= \sum_{u \in \mathcal{U}_i^t} (f_{ui}^{\alpha}(t)(r_{ui} - \bar{r}_u(t)))^2 = \sum_{u \in \Delta\mathcal{U}_i^t} (r_{ui} - \bar{r}_u(t))^2 + \sum_{u \in \mathcal{U}_i^{t-1}} (f_{ui}^{\alpha}(t)(r_{ui} - \bar{r}_u(t)))^2 \\
&= \Delta Q_i(t) + e^{-2\alpha} \sum_{u \in \mathcal{U}_i^{t-1}} (f_{ui}^{\alpha}(t-1)(r_{ui} - \bar{r}_u(t-1) - \epsilon(t)))^2 \\
&= \Delta Q_i(t) + e^{-2\alpha} \sum_{u \in \mathcal{U}_i^{t-1}} (f_{ui}^{\alpha}(t-1)(r_{ui} - \bar{r}_u(t-1)))^2 - 2e^{-2\alpha} \sum_{u \in \mathcal{U}_i^{t-1}} \epsilon(t) \cdot f_{ui}^{2\alpha}(t-1)(r_{ui} - \bar{r}_u(t-1)) \\
&\quad + e^{-2\alpha} \sum_{u \in \mathcal{U}_i^{t-1}} \epsilon(t)^2 \cdot f_{ui}^{2\alpha}(t-1) \\
&= \Delta Q_i(t) + e^{-2\alpha} \sum_{u \in \mathcal{U}_i^{t-1}} (f_{ui}^{\alpha}(t-1)(r_{ui} - \bar{r}_u(t-1)))^2 \\
&\quad - 2e^{-2\alpha} \sum_{u \in \mathcal{U}_i^{t-1}} (\epsilon(t-1) + \Delta\epsilon) \cdot f_{ui}^{2\alpha}(t-1)(r_{ui} - \bar{r}_u(t-1)) + e^{-2\alpha} \sum_{u \in \mathcal{U}_i^{t-1}} (\epsilon(t-1) + \Delta\epsilon)^2 \cdot f_{ui}^{2\alpha}(t-1)
\end{aligned}$$

Ignoring negligibly small higher order terms with multiplicative factor $\Delta\epsilon \cdot f_{ui}^{2\alpha(t)}$ as each of the terms $\{\Delta\epsilon, f_{ui}^{\alpha}(t)\} << 1$, we get the following:

$$\begin{aligned}
Q_i(t) = \Delta Q_i(t) &+ e^{-2\alpha} \sum_{u \in \mathcal{U}_i^{t-1}} (f_{ui}^{\alpha}(t-1)(r_{ui} - \bar{r}_u(t-1)))^2 \\
&- 2e^{-2\alpha} \sum_{u \in \mathcal{U}_i^{t-1}} \epsilon(t-1) \cdot f_{ui}^{2\alpha}(t-1)(r_{ui} - \bar{r}_u(t-1)) + e^{-2\alpha} \sum_{u \in \mathcal{U}_i^{t-1}} \epsilon(t-1)^2 \cdot f_{ui}^{2\alpha}(t-1)
\end{aligned}$$

We rewrite this expression for $Q_i(t)$ in the following manner.

$$Q_i(t) = \Delta Q_i(t) + e^{-2\alpha}[Q_i(t-1) - L_i(t-1) + M_i(t-1)]$$

Interestingly, the terms required for incrementally updating $Q_i(t)$, namely $\Delta Q_i(t)$, $\Delta L_{ij}(t)$ and $\Delta M_{ij}(t)$, have a time complexity bounded by $\mathcal{O}(|\Delta\mathcal{U}_i^t|)$. Note that the complexity for the non-incremental update is again $\mathcal{O}(|\mathcal{U}_i^t|)$. □

Hence, the final incremental relations for adjusted cosine similarity are as follows.

$$P_{ij}(t) = \Delta P_{ij}(t) + e^{-2\alpha}[P_{ij}(t-1) - L_{ij}(t-1) + M_{ij}(t-1)] \tag{8.4}$$

$$Q_i(t) = \Delta Q_i(t) + e^{-2\alpha}[Q_i(t-1) - L_i(t-1) + M_i(t-1)] \tag{8.5}$$

$$L_{ij}(t) = \Delta L_{ij}(t) + e^{-2\alpha}[L_{ij}(t-1) - 2M_{ij}(t-1)] \tag{8.6}$$

$$M_{ij}(t) = \Delta M_{ij}(t) + e^{-2\alpha} M_{ij}(t-1) \tag{8.7}$$

The I-SIM values (S_{ij}) can thus be computed on-the-fly, leveraging the incrementally updated $P_{ij}(t)$ and $Q_i(t)$ values. We only need to store the P, L, M and Q values which requires $\mathcal{O}(|\mathcal{I}|^2)$ space. Unlike classical non-incremental algorithms [157], we require extra storage for the adjustment terms (L, M). Note that the non-incremental algorithm would also require $\mathcal{O}(|\mathcal{I}|^2)$ space for storing the item-item similarities. Nonetheless, incremental as well as non-incremental algorithms could benefit from sparse data structures for significantly reducing the storage requirements.

Ignoring the higher order terms mentioned throughout the proofs does not pose a limitation to I-SIM. Additional levels of adjustment terms (similar to L, M) could be employed to overcome these approximations at the cost of increasing the storage requirements (the space complexity remains $\mathcal{O}(|\mathcal{I}|^2)$). Nevertheless, as we also demonstrate empirically (§4.1.4), these negligibly small higher order terms indeed do not impact our accuracy. Approximate similarity computations have been successfully used to provide performance benefits, both in terms of computation time and storage with negligible impact on the accuracy [30, 140, 7]. Therefore, since there is no practical trade-off between accuracy and storage, we choose to employ only a single level of adjustment terms.

8.2 Appendix B: X-Map

8.2.1 Correctness proofs

Theorem 4 (Privacy guarantee)**.** *Given any item i, we denote the global sensitivity of* X-Sim *by GS and the similarity between i and any arbitrary item j by* X-Sim(i,j)*. Our Private Replacement Selection (PRS) mechanism, which outputs j as the replacement with a probability proportional to $exp(\frac{\epsilon \cdot \text{X-SIM}(i,j)}{2 \cdot GS})$, ensures ϵ-differential privacy.*

Proof. Consider two datasets D and D' which differ at one user, say u. We denote X-Sim (i,j) in dataset D as $q(D,i,j)$ and $I(i)$ as the set of items in target domain with quantified X-Sim values. The global sensitivity (GS) is defined as $max_{D,D'}||q(D,i,j)-q(D',i,j)||_1$. Our PRS mechanism outputs an item j as a private replacement for i. Then, we get the following equality:

$$\frac{Pr[PRS(i,I(i),q(D,I(i)))=j]}{Pr[PRS(i,I(i),q(D',I(i)))=j]} = \frac{exp(\frac{\epsilon \cdot q(D,i,j)}{2 \cdot GS})}{\sum\limits_{k \in I(i)} exp(\frac{\epsilon \cdot q(D,i,k)}{2 \cdot GS})} \div \frac{exp(\frac{\epsilon \cdot q(D',i,j)}{2 \cdot GS})}{\sum\limits_{k \in I(i)} exp(\frac{\epsilon \cdot q(D',i,k)}{2 \cdot GS})}$$

$$= \underbrace{\frac{exp(\frac{\epsilon \cdot q(D,i,j)}{2 \cdot GS})}{exp(\frac{\epsilon \cdot q(D',i,j)}{2 \cdot GS})}}_{\text{P}} \cdot \underbrace{\frac{\sum\limits_{k \in I(i)} exp(\frac{\epsilon \cdot q(D',i,k)}{2 \cdot GS})}{\sum\limits_{k \in I(i)} exp(\frac{\epsilon \cdot q(D,i,k)}{2 \cdot GS})}}_{\text{Q}}$$

$$P = exp(\frac{\epsilon \cdot (q(D,i,j) - q(D',i,j))}{2 \cdot GS}) \leq exp(\frac{\epsilon \cdot GS}{2 \cdot GS}) = exp(\frac{\epsilon}{2})$$

$$Q = \frac{\sum\limits_{k \in I(i)} exp(\frac{\epsilon \cdot q(D',i,k)}{2 \cdot GS})}{\sum\limits_{k \in I(i)} exp(\frac{\epsilon \cdot q(D,i,k)}{2 \cdot GS})} \leq \frac{\sum\limits_{k \in I(i)} exp(\frac{\epsilon \cdot (q(D,i,k)+GS)}{2 \cdot GS})}{\sum\limits_{k \in I(i)} exp(\frac{\epsilon \cdot q(D,i,k)}{2 \cdot GS})} = \frac{exp(\frac{\epsilon}{2}) \cdot \sum\limits_{k \in I(i)} exp(\frac{\epsilon \cdot q(D,i,k)}{2 \cdot GS})}{\sum\limits_{k \in I(i)} exp(\frac{\epsilon \cdot q(D,i,k)}{2 \cdot GS})} = exp(\frac{\epsilon}{2})$$

Therefore, we get the following inequality:

$$\frac{Pr[PRS(i,I(i),q(D,I(i)))=j]}{Pr[PRS(i,I(i),q(D',I(i)))=j]} \leq exp(\epsilon)$$

Hence, PRS provides ϵ-differential privacy. □

Theorem 5 (Similarity-based sensitivity)**.** *Given any score function $q : \mathcal{R} \rightarrow R$ and a dataset D, we formulate the similarity-based sensitivity corresponding to a score function $q_i(I,j)$ for a pair of items i and j as:*

$$SS(i,j) = max\{max_{u_x \in \mathcal{U}_{ij}}(\frac{r_{xi} \times r_{xj}}{\parallel r'_i \parallel \times \parallel r'_j \parallel}), max_{u_x \in \mathcal{U}_{ij}}(\frac{r_i \cdot r_j}{\parallel r'_i \parallel \times \parallel r'_j \parallel} - \frac{r_i \cdot r_j}{\parallel r_i \parallel \times \parallel r_j \parallel})\}$$

Proof. We now provide the proof of the similarity-based sensitivity. First, we define similarity-based sensitivity (SS) as follows.

$$SS(i,j) = max \parallel s(i,j) - s'(i,j) \parallel_1$$

We then insert the similarity values for $s(i,j)$. A rating vector $r_i = [r_{ai}, ..., r_{xi}, r_{yi}]$ consists of all the ratings for an item i. Note that here a rating r_{xi} denotes the result after subtracting the average rating of user x ($\bar{r_x}$) from the actual rating provide by x for an item i. Then, we get the following equality:

$$\begin{aligned} s(i,j) - s'(i,j) &= \frac{r_i \cdot r_j}{\parallel r_i \parallel \times \parallel r_j \parallel} - \frac{r'_i \cdot r'_j}{\parallel r'_i \parallel \times \parallel r'_j \parallel} \\ &= \frac{r_i \cdot r_j \times \parallel r'_i \parallel \times \parallel r'_j \parallel - r'_i \cdot r'_j \times \parallel r_i \parallel \times \parallel r_j \parallel}{\parallel r_i \parallel \times \parallel r_j \parallel \times \parallel r'_i \parallel \times \parallel r'_j \parallel} = \frac{P}{Q} \end{aligned}$$

We assume that the profile of a user x, in D, is not present in D'. This user rated both i and j in D. Note that if this user rated one of these items or none, then the similarity value does not depend on the presence or absence of this user in the dataset. Hence, the following inequality holds: $\parallel r'_i \parallel \times \parallel r'_j \parallel \leq \parallel r_i \parallel \times \parallel r_j \parallel$.

Based on our assumption, P= $(r_i \cdot r_j \times \parallel r'_i \parallel \times \parallel r'_j \parallel - r'_i \cdot r'_j \times \parallel r_i \parallel \times \parallel r_j \parallel)$ and Q=$(\parallel r_i \parallel \times \parallel r_j \parallel \times \parallel r'_i \parallel \times \parallel r'_j \parallel)$. Hence, $Q \geq 0$ and depending on whether $P \geq 0$ or $P \leq 0$ we have two conditions which are as follows.

If $P \geq 0$, then we get the following inequality:

$$\begin{aligned} \parallel s(i,j) - s'(i,j) \parallel_1 &= \frac{r_i \cdot r_j \times \parallel r'_i \parallel \times \parallel r'_j \parallel - r'_i \cdot r'_j \times \parallel r_i \parallel \times \parallel r_j \parallel}{\parallel r_i \parallel \times \parallel r_j \parallel \times \parallel r'_i \parallel \times \parallel r'_j \parallel} \\ &\leq \frac{(r_i \cdot r_j - r'_i \cdot r'_j) \times \parallel r_i \parallel \times \parallel r_j \parallel}{\parallel r_i \parallel \times \parallel r_j \parallel \times \parallel r'_i \parallel \times \parallel r'_j \parallel} = \frac{(r_i \cdot r_j - r'_i \cdot r'_j)}{\parallel r'_i \parallel \times \parallel r'_j \parallel} \end{aligned}$$

If $P \leq 0$, then we get the following inequality:

$$\begin{aligned} \parallel s(i,j) - s'(i,j) \parallel_1 &= \frac{r'_i \cdot r'_j \times \parallel r_i \parallel \times \parallel r_j \parallel - r_i \cdot r_j \times \parallel r'_i \parallel \times \parallel r'_j \parallel}{\parallel r_i \parallel \times \parallel r_j \parallel \times \parallel r'_i \parallel \times \parallel r'_j \parallel} \\ &= \frac{(r_i \cdot r_j - r_{xi} \times r_{xj}) \times \parallel r_i \parallel \times \parallel r_j \parallel}{\parallel r_i \parallel \times \parallel r_j \parallel \times \parallel r'_i \parallel \times \parallel r'_j \parallel} - \frac{r_i \cdot r_j \times \parallel r'_i \parallel \times \parallel r'_j \parallel}{\parallel r_i \parallel \times \parallel r_j \parallel \times \parallel r'_i \parallel \times \parallel r'_j \parallel} \\ &= \frac{r_i \cdot r_j \times (\parallel r_i \parallel \times \parallel r_j \parallel - \parallel r'_i \parallel \times \parallel r'_j \parallel)}{\parallel r_i \parallel \times \parallel r_j \parallel \times \parallel r'_i \parallel \times \parallel r'_j \parallel} - \frac{r_{xi} \times r_{xj} \times \parallel r_i \parallel \times \parallel r_j \parallel}{\parallel r_i \parallel \times \parallel r_j \parallel \times \parallel r'_i \parallel \times \parallel r'_j \parallel} \end{aligned}$$

$$\leq \frac{r_i \cdot r_j \times (\parallel r_i \parallel \times \parallel r_j \parallel - \parallel r'_i \parallel \times \parallel r'_j \parallel)}{\parallel r_i \parallel \times \parallel r_j \parallel \times \parallel r'_i \parallel \times \parallel r'_j \parallel} = \frac{r_i \cdot r_j}{\parallel r'_i \parallel \times \parallel r'_j \parallel} - \frac{r_i \cdot r_j}{\parallel r_i \parallel \times \parallel r_j \parallel}$$

Hence, the similarity-based sensitivity is as follows:

$$SS(i,j) = max\{max_{u_x \in \mathcal{U}_{ij}}(\frac{r_{xi} \times r_{xj}}{\parallel r'_i \parallel \times \parallel r'_j \parallel}), max_{u_x \in \mathcal{U}_{ij}}(\frac{r_i \cdot r_j}{\parallel r'_i \parallel \times \parallel r'_j \parallel} - \frac{r_i \cdot r_j}{\parallel r_i \parallel \times \parallel r_j \parallel})\}$$

□

8.2.2 Additional experiments

A. User-based vs Item-based recommenders

Different practical deployment scenarios benefit from the proper choice of the recommendation algorithm. One requirement, which is crucial to any deployment scenario, is *Scalability*. We highlight below two factors which affect scalability in such deployment scenarios.

- Item-based recommenders leverage item-item similarities whereas user-based recommenders leverage user-user similarities. For big e-commerce players (e.g., Amazon, e-Bay), the number of items is significantly less than the number of users. Hence, such players would prefer an item-based approach for scalability purpose. For new players, the number of items would be significantly larger than the number of users. Such new players would thus benefit from a user-based approach for scalability.
- Similarities between items tend not to vary much from day to day, or even week to week [5]. Over ranges of months, however, the similarities do vary due to various temporal factors like item popularity, behavioral drift of users. In this sense, item-item similarities are much less dynamic than user-user similarities and thus they require fewer updates.

We conducted an experiment, which we describe below, through which we demonstrate how the computation time differs for these two algorithms in two deployment scenarios. In both the scenarios, we consider the movies domain as the source domain and the books domain as the target domain.

S_1. In the first deployment scenario, we retain the original Amazon dataset. The movies dataset consists of ratings from 473,764 users for 128,402 movies whereas the books dataset consists of ratings from 725,846 users for 403,234 books. We observe that the number of users is approximately 1.8× the number of books in the target domain. This deployment scenario depicts the instance of big e-commerce players.

S_2. In the second deployment scenario, we modify the dataset of the target domain (books). The profiles of the overlapping users are retained unchanged whereas those of the non-overlapping users in the target domain are sorted, in a descending order, by the number of corresponding ratings in the profiles (profile size). Finally, only the top 100,000 users are retained in the target domain. This customized dataset consists of 104,535 users and 236,710 books in the target domain. We observe that the number of items is now nearly 2.27× the number of users. This deployment scenario depicts the instance of new e-commerce players.

Approach	S_1	S_2
	Time (s)	Time (s)
X-MAP-UB	886	**870**
X-MAP-IB	**844**	962
NX-MAP-UB	822	**805**
NX-MAP-IB	**674**	877

Table 8.1 – *Comparison between user-based (*UB*) and item-based (*IB*) recommenders in different deployment scenarios with Amazon datasets. Bold denotes faster computation time relative to the alternative.*

We evaluate the recommendation quality in terms of Mean Absolute Error (MAE). We observe the following behaviour from Table 8.1.

- The item-based version (IB) is computationally faster than the user-based alternative (UB) in scenario S_1 where the number of users is approximately 1.8× the number of books in the target domain.
- The user-based version (UB) is computationally faster than the item-based alternative (IB) in scenario S_2 where the number of items is nearly 2.27× the number of users.

B. Comparison with a dimensionality reduction approach

We now compare X-MAP with a dimensionality reduction approach such as matrix factorization. For this purpose, we choose Spark's Alternating Least Squares (ALS) implementation available with its MLLIB library, denoted here by MLLIB-ALS, and apply it over the combined Amazon dataset (movies, books) of items and users while keeping the test set same as the one used for evaluating X-MAP (mentioned in the paper). We optimally tune MLLIB-ALS with varying parameters like the number of latent factors in the model (rank) or the regularization parameter (λ) to obtain the best recommendation quality.

	S:Movie, T:Book	S:Book, T:Movie
NX-MAP	0.5332	0.5470
X-MAP	0.6616	0.6884
MLLIB-ALS	0.7527	0.8237

Table 8.2 – *MAE comparison between* NX-MAP*,* X-MAP *and* MLLIB-ALS *on Amazon datasets.*

Table 8.2 depicts the results of this experiment. We observe that MLLIB-ALS does not perform so well in a heterogeneous recommendation scenario which could be partially attributed to the decreased density [1] of the combined Amazon dataset (movies and books), shown in Table 8.3, as well as the different online behavior of the users in the two domains.

Books	Movies	Books+Movies
0.0204 %	0.0569 %	**0.0147 %**

Table 8.3 – *Densities for two domains in the Amazon dataset.*

---

[1] Rating density is defined as the fraction of collected ratings over all the possible ratings.

Bibliography

[1] M. S. Ackerman and D. T. Davis Jr. Privacy and security issues in e-commerce. In *New Economy Handbook*, pages 911–930. Academic Press/ Elsevier, 2003. [Cited on page 96]

[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. In *Transactions on Knowledge and Data Engineering (TKDE)*, pages 734–749. IEEE, 2005. [Cited on pages 3, 5, 60, 101, and 113]

[3] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 29–42. ACM, 2013. [Cited on page 24]

[4] N. Ahituv, Y. Lapid, and S. Neumann. Processing encrypted data. *Communications of the ACM*, 30(9):777–780, 1987. [Cited on page 97]

[5] K. Ali and W. Van Stam. Tivo: making show recommendations using a distributed collaborative filtering architecture. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 394–401. ACM, 2004. [Cited on pages xvii, 33, 44, 45, and 142]

[6] D. Alistarh, J. Li, R. Tomioka, and M. Vojnovic. Qsgd: Randomized quantization for communication-optimal stochastic gradient descent. In *NIPS*, 2017. [Cited on page 131]

[7] G. Amato and P. Savino. Approximate similarity search in metric spaces using inverted files. In *Proceedings of the 3rd international conference on Scalable information systems*, page 28. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008. [Cited on page 139]

[8] X. Amatriain, J. M. Pujol, and N. Oliver. I like it... i like it not: Evaluating user ratings noise in recommender systems. In *International Conference on User Modeling, Adaptation, and Personalization*, pages 247–258. Springer, 2009. [Cited on page 124]

[9] Aws ec2 instances and pricing. http://aws.amazon.com/ec2. [Cited on page 32]

[10] Amazon: About recommendations. https://www.amazon.com/gp/help/customer/display.html?ie=UTF8&nodeId=16465251. [Cited on page 76]

[11] L. Ardissono, A. Goy, G. Petrone, and M. Segnan. A multi-agent infrastructure for developing personalized web-based systems. *ACM Transactions on Internet Technology (TOIT)*, 5(1):47–69, 2005. [Cited on page 33]

[12] U. Awada, K. Li, and Y. Shen. Energy consumption in cloud computing data centers. *International Journal of Cloud Computing and services science*, 3(3):145, 2014. [Cited on page 39]

[13] L. Baltrunas and X. Amatriain. Towards time-dependant recommendation based on implicit feedback. In *Workshop on context-aware recommender systems (CARS'09)*, 2009. [Cited on page 76]

[14] L. Baltrunas and F. Ricci. Context-based splitting of item ratings in collaborative filtering. In *Proceedings of the third ACM conference on Recommender systems*, pages 245–248. ACM, 2009. [Cited on pages 103, 118, and 125]

[15] R. Bambini, P. Cremonesi, and R. Turrin. A recommender system for an iptv service provider: a real large-scale production environment. *Recommender systems handbook*, pages 299–331, 2011. [Cited on page 39]

[16] O. Barkan and N. Koenigstein. Item2vec: neural item embedding for collaborative filtering. In *Machine Learning for Signal Processing (MLSP), 2016 IEEE 26th International Workshop on*, pages 1–6. IEEE, 2016. [Cited on pages 60, 67, and 75]

[17] L. Barkhuus and A. K. Dey. Location-based services for mobile telephony: a study of users' privacy concerns. In *Interact*, volume 3, pages 702–712, 2003. [Cited on page 76]

[18] A. Ben-David, N. Nisan, and B. Pinkas. Fairplaymp: a system for secure multi-party computation. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 257–266. ACM, 2008. [Cited on page 98]

[19] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003. [Cited on page 67]

[20] S. Berkovsky, T. Kuflik, and F. Ricci. Cross-domain mediation in collaborative filtering. *User Modeling 2007*, pages 355–359, 2007. [Cited on pages 103, 118, 124, and 125]

[21] S. Berkovsky, T. Kuflik, and F. Ricci. Distributed collaborative filtering with domain specialization. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 33–40. ACM, 2007. [Cited on page 124]

[22] M. Bertier, D. Frey, R. Guerraoui, A.-M. Kermarrec, and V. Leroy. The gossple anonymous social network. In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, pages 191–211. Springer-Verlag, 2010. [Cited on pages 24 and 25]

[23] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008. [Cited on page 61]

[24] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E. Talbi, and I. Touche. Grid'5000: A large scale and highly reconfigurable experimental grid testbed. In *HPCA*, 2006. [Cited on page 31]

[25] M. Bonett. Personalization of web services: opportunities and challenges. *Ariadne*, 28, 2001. [Cited on page 1]

[26] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *NDSS*, 2015. [Cited on page 130]

[27] A. Boutet, D. Frey, R. Guerraoui, A.-M. Kermarrec, and R. Patra. Hyrec: Leveraging browsers for scalable recommenders. In *Proceedings of the 15th International Middleware Conference*, pages 85–96. ACM, 2014. [Cited on pages 4, 65, 81, and 86]

[28] M. Brand. Fast online svd revisions for lightweight recommender systems. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 37–46. SIAM, 2003. [Cited on page 2]

[29] A. Brenner, B. Pradel, N. Usunier, and P. Gallinari. Predicting most rated items in weekly recommendation with temporal regression. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*, pages 24–27. ACM, 2010. [Cited on page 15]

[30] M. Bury and C. Schwiegelshohn. Efficient similarity search in dynamic data streams. *arXiv preprint arXiv:1605.03949*, 2016. [Cited on page 139]

[31] J. A. Calandrino, A. Kilzer, A. Narayanan, E. W. Felten, and V. Shmatikov. " you might also like:" privacy risks of collaborative filtering. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 231–246. IEEE, 2011. [Cited on page 5]

[32] P. G. Campos, A. Bellogín, F. Díez, and J. E. Chavarriaga. Simple time-biased knn-based recommendations. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*, pages 20–23. ACM, 2010. [Cited on pages 59 and 76]

[33] J. Canny. Collaborative filtering with privacy. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 45–57. IEEE, 2002. [Cited on page 98]

[34] I. Cantador, I. Fernández-Tobías, and A. Bellogín. Relating personality types with user preferences in multiple entertainment domains. In *CEUR Workshop Proceedings*. Shlomo Berkovsky, 2013. [Cited on page 125]

[35] F. Carmagnola and F. Cena. User identification for cross-system personalisation. *Information Sciences*, 179(1):16–32, 2009. [Cited on pages 16 and 103]

[36] F. Carmagnola, F. Cena, and C. Gena. User model interoperability: a survey. *User Modeling and User-Adapted Interaction*, 21(3):285–331, 2011. [Cited on pages 16 and 103]

[37] C. Chen, H. Yin, J. Yao, and B. Cui. Terec: A temporal recommender system over tweet stream. *Proceedings of the VLDB Endowment*, 6(12):1254–1257, 2013. [Cited on pages 59 and 68]

[38] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz. Energy efficiency for large-scale mapreduce workloads with significant interactive analysis. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 43–56. ACM, 2012. [Cited on page 2]

[39] E. Christakopoulou and G. Karypis. Hoslim: higher-order sparse linear method for top-n recommender systems. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 38–49. Springer, 2014. [Cited on pages 5, 60, and 75]

[40] Ciao. http://www.ciao.com/. [Cited on pages 62 and 71]

[41] G. Cormode and M. Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment*, 1(2):1530–1541, 2008. [Cited on page 45]

[42] N. Craswell and M. Szummer. Random walks on the click graph. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 239–246. ACM, 2007. [Cited on page 75]

[43] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46. ACM, 2010. [Cited on pages 17, 71, and 73]

[44] P. Cremonesi, A. Tripodi, and R. Turrin. Cross-domain recommender systems. In *Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on*, pages 496–503. Ieee, 2011. [Cited on pages 101, 102, 103, 117, and 125]

[45] G. Damaskinos, R. Guerraoui, and R. Patra. Capturing the moment: Lightweight similarity computations. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pages 747–758. IEEE, 2017. [Cited on pages 2 and 5]

[46] A. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW*, 2007. [Cited on pages 2, 23, and 28]

[47] A. de Spindler, M. C. Norrie, M. Grossniklaus, and B. Signer. Spatio-temporal proximity as a basis for collaborative filtering in mobile environments. In *CAISE*, pages 912–926, 2006. [Cited on page 76]

[48] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012. [Cited on pages 68 and 70]

[49] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 2008. [Cited on pages 2 and 23]

[50] Deepdist: Lightning-fast deep learning on spark. http://deepdist.com/. [Cited on pages 68 and 70]

[51] E. Diaz-Aviles, L. Drumond, L. Schmidt-Thieme, and W. Nejdl. Real-time top-n recommendation in social streams. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 59–66. ACM, 2012. [Cited on page 2]

[52] Digg. http://digg.com. [Cited on page 28]

[53] Y. Ding and X. Li. Time weight collaborative filtering. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 485–492. ACM, 2005. [Cited on pages 6, 59, 103, and 110]

[54] W. Dong, C. Moses, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*, pages 577–586. ACM, 2011. [Cited on pages 2, 24, and 26]

[55] C. Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008. [Cited on pages 5, 15, 34, 79, 81, 95, and 96]

[56] C. Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54(1):86–95, 2011. [Cited on pages 98 and 117]

[57] C. Dwork and J. Lei. Differential privacy and robust statistics. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 371–380. ACM, 2009. [Cited on pages 5, 15, 79, 81, 95, and 96]

[58] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, volume 3876, pages 265–284. Springer, 2006. [Cited on page 15]

[59] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Theoretical Computer Science*, 9(3-4):211–407, 2013. [Cited on page 15]

[60] M. D. Ekstrand, J. T. Riedl, J. A. Konstan, et al. Collaborative filtering recommender systems. *Foundations and Trends® in Human–Computer Interaction*, 4(2):81–173, 2011. [Cited on pages 12 and 23]

[61] A. Ene, S. Im, and B. Moseley. Fast clustering using mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 681–689. ACM, 2011. [Cited on page 2]

[62] Z. Erkin, M. Beye, T. Veugen, and R. L. Lagendijk. Efficiently computing private recommendations. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5864–5867. IEEE, 2011. [Cited on page 98]

[63] Ó. Fontenla-Romero, B. Guijarro-Berdiñas, D. Martinez-Rego, B. Pérez-Sánchez, and D. Peteiro-Barral. Online machine learning. *Efficiency and Scalability Methods for Computational Intellect*, 27, 2013. [Cited on page 68]

[64] A. Friedman and A. Schuster. Data mining with differential privacy. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 493–502. ACM, 2010. [Cited on pages 15, 81, 95, and 96]

[65] M. Ge, C. Delgado-Battenfeld, and D. Jannach. Beyond accuracy: evaluating recommender systems by coverage and serendipity. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 257–260. ACM, 2010. [Cited on pages 80, 88, and 94]

[66] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 69–77. ACM, 2011. [Cited on page 2]

[67] J. Golbeck, B. Parsia, and J. Hendler. Trust networks on the semantic web. In *International Workshop on Cooperative Information Agents*, pages 238–249. Springer, 2003. [Cited on page 17]

[68] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992. [Cited on page 74]

[69] G. González, B. López, and J. L. de la Rosa. A multi-agent smart user model for cross-domain recommender systems. *Proceedings of Beyond Personalization*, 2005. [Cited on page 124]

[70] S. Gordea and M. Zanker. Time filtering for better recommendations with small and sparse rating matrices. *Web Information Systems Engineering–WISE 2007*, pages 171–183, 2007. [Cited on page 76]

[71] S. Gorman. Nsa.'s domestic spying grows as agency sweeps up data. *The Wall Street Journal*, 10, 2008. [Cited on page 97]

[72] T. Graepel, K. Lauter, and M. Naehrig. Ml confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptology*, pages 1–21. Springer, 2012. [Cited on page 130]

[73] Grid5000. https://www.grid5000.fr/. [Cited on page 71]

[74] R. Grover and M. J. Carey. Extending map-reduce for efficient predicate-based sampling. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 486–497. IEEE, 2012. [Cited on page 2]

[75] R. Guerraoui, A.-M. Kermarrec, T. Lin, and R. Patra. Heterogeneous recommendations: what you might like to read after watching interstellar. *Proceedings of the VLDB Endowment*, 10(10):1070–1081, 2017. [Cited on page 6]

[76] R. Guerraoui, A.-M. Kermarrec, R. Patra, and M. Taziki. D2p: Distance-based differential privacy in recommenders. *PVLDB*, 8(8), 2015. [Cited on pages 5, 83, and 87]

[77] R. Guerraoui, A.-M. Kermarrec, R. Patra, M. Valiyev, and J. Wang. I know nothing about you but here is what you might like. In *Dependable Systems and Networks (DSN), 2017 47th Annual IEEE/IFIP International Conference on*, pages 439–450. IEEE, 2017. [Cited on pages 6 and 98]

[78] R. Guerraoui, E. L. Merrer, R. Patra, and J.-R. Vigouroux. Sequences, items and latent links: Recommendation with consumed item packs. *arXiv preprint arXiv:1711.06100*, 2017. [Cited on page 5]

[79] A. Gulin, I. Kuralenok, and D. Pavlov. Winning the transfer learning track of yahoo!'s learning to rank challenge with yetirank. In *Proceedings of the Learning to Rank Challenge*, pages 63–76, 2011. [Cited on page 125]

[80] Hadoop. http://hadoop.apache.org/. [Cited on page 31]

[81] S. Harizopoulos, M. A. Shah, J. Meza, and P. Ranganathan. Energy efficiency: The new holy grail of data management systems research. In *CIDR*, 2009. [Cited on page 59]

[82] R. He and J. McAuley. Fusing similarity models with markov chains for sparse sequential recommendation. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pages 191–200. IEEE, 2016. [Cited on pages 72 and 76]

[83] J. Herlocker, J. A. Konstan, and J. Riedl. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Information retrieval*, 5(4):287–310, 2002. [Cited on pages 11 and 59]

[84] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237. ACM, 1999. [Cited on pages 6, 103, and 117]

[85] 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. [Cited on page 94]

[86] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. [Cited on page 131]

[87] T. R. Hoens, M. Blanton, and N. V. Chawla. A private and reliable recommendation system for social networks. In *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, pages 816–825. IEEE, 2010. [Cited on page 98]

[88] T. Hofmann and D. Hartmann. Collaborative filtering with privacy via factor analysis. In *Proceedings of the 2005 ACM symposium on applied computing*, pages 791–795, 2005. [Cited on page 34]

[89] T. Hofmann and J. Puzicha. Latent class models for collaborative filtering. In *IJCAI*, volume 99, 1999. [Cited on page 11]

[90] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. Ieee, 2008. [Cited on page 75]

[91] C.-K. Huang, L.-F. Chien, and Y.-J. Oyang. Relevant term suggestion in interactive web search based on contextual information in query session logs. *Journal of the American Society for Information Science and Technology*, 54(7):638–649, 2003. [Cited on page 75]

[92] Y. Huang, B. Cui, W. Zhang, J. Jiang, and Y. Xu. Tencentrec: Real-time stream recommendation in practice. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 227–238. ACM, 2015. [Cited on pages 2, 4, 11, 39, 40, 41, 47, and 59]

[93] P. Jain, P. Kothari, and A. Thakurta. Differentially private online learning. In *Conference on Learning Theory*, pages 1–24, 2012. [Cited on page 96]

[94] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender systems: an introduction.* Cambridge University Press, 2010. [Cited on pages 48 and 121]

[95] G. Jawaheer, M. Szomszor, and P. Kostkova. Comparison of implicit and explicit feedback from an online music recommendation service. In *proceedings of the 1st international workshop on information heterogeneity and fusion in recommender systems*, pages 47–51. ACM, 2010. [Cited on page 75]

[96] *Jester.* http://goldberg.berkeley.edu/jester-data/. [Cited on page 89]

[97] J. Jiang, B. Cui, C. Zhang, and L. Yu. Heterogeneity-aware distributed parameter servers. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 463–478. ACM, 2017. [Cited on pages 4 and 35]

[98] B. Joshi. Multithreading in web pages using web workers. In *HTML5 Programming for ASP. NET Developers*, pages 255–275. Springer, 2012. [Cited on page 34]

[99] S. Kabbur, X. Ning, and G. Karypis. Fism: factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 659–667. ACM, 2013. [Cited on pages 41, 53, and 59]

[100] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011. [Cited on page 95]

[101] S. P. Kasiviswanathan, K. Nissim, and H. Jin. Private incremental regression. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 167–182. ACM, 2017. [Cited on page 131]

[102] S. Katzenbeisser and M. Petkovic. Privacy-preserving recommendation systems for consumer healthcare services. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pages 889–895. IEEE, 2008. [Cited on page 96]

[103] G. Kellaris, S. Papadopoulos, X. Xiao, and D. Papadias. Differentially private event sequences over infinite streams. *Proceedings of the VLDB Endowment*, 7(12):1155–1166, 2014. [Cited on page 130]

[104] R. Khoussainov, X. Zuo, and N. Kushmerick. Grid-enabled weka: A toolkit for machine learning on the grid. *ERCIM news*, 59:47–48, 2004. [Cited on page 33]

[105] B. Kille, A. Lommatzsch, R. Turrin, A. Serény, M. Larson, T. Brodt, J. Seiler, and F. Hopfgartner. Overview of clef newsreel 2015: News recommendation evaluation lab. 2015. [Cited on page 52]

[106] B. Kille, A. Lommatzsch, R. Turrin, A. Serény, M. Larson, T. Brodt, J. Seiler, and F. Hopfgartner. Stream-based recommendations: Online and offline evaluation as a service. In *International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 497–517. Springer, 2015. [Cited on page 51]

[107] J. A. Konstan and J. Riedl. Recommender systems: from algorithms to user experience. *User Modeling and User-Adapted Interaction*, 22(1):101–123, 2012. [Cited on page 1]

[108] H. J. Kook. Profiling multiple domains of user interests and using them for personalized web support. In *International Conference on Intelligent Computing*, pages 512–520. Springer, 2005. [Cited on page 124]

[109] J. Koomey. Growth in data center electricity use 2005 to 2010. *A report by Analytical Press, completed at the request of The New York Times*, 9, 2011. [Cited on page 59]

[110] Y. Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97, 2010. [Cited on pages 13, 39, 41, 53, and 59]

[111] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009. [Cited on pages 41, 72, 76, and 87]

[112] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. [Cited on page 37]

[113] N. Lathia, S. Hailes, and L. Capra. Temporal collaborative filtering with adaptive neighbourhoods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 796–797. ACM, 2009. [Cited on page 59]

[114] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1188–1196, 2014. [Cited on page 125]

[115] T. Q. Lee, Y. Park, and Y.-T. Park. An empirical study on effectiveness of temporal information as implicit ratings. *Expert systems with Applications*, 36(2):1315–1321, 2009. [Cited on pages 5, 72, and 76]

[116] D. Lemire and A. Maclachlan. Slope one predictors for online rating-based collaborative filtering. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 471–475. SIAM, 2005. [Cited on page 118]

[117] K. Lerman and T. Hogg. Using a model of social dynamics to predict popularity of news. In *Proceedings of the 19th international conference on World wide web*, pages 621–630. ACM, 2010. [Cited on page 39]

[118] J. J. Levandoski, M. D. Ekstrand, M. J. Ludwig, A. Eldawy, M. F. Mokbel, and J. T. Riedl. Recbench: benchmarks for evaluating performance of recommender system architectures. *Proceedings of the VLDB Endowment*, 4(11), 2011. [Cited on page 39]

[119] T. Li and T. Unger. Willing to pay for quality personalization? trade-off between quality and privacy. *European Journal of Information Systems*, 21(6):621–642, 2012. [Cited on page 79]

[120] X. Lian, Y. Huang, Y. Li, and J. Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745, 2015. [Cited on pages 4, 16, and 35]

[121] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. In *Internet Computing*, pages 76–80. IEEE, 2003. [Cited on page 23]

[122] N. N. Liu, M. Zhao, E. Xiang, and Q. Yang. Online evolutionary collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 95–102. ACM, 2010. [Cited on pages 5, 13, 39, 40, 51, 59, and 60]

[123] Location privacy via pictures. https://www.eff.org/deeplinks/2012/04/picture-worth-thousand-words-including-your-location. [Cited on page 130]

[124] A. Loizou. *How to recommend music to film buffs: enabling the provision of recommendations from multiple domains*. PhD thesis, University of Southampton, 2009. [Cited on page 124]

[125] B. Loni, Y. Shi, M. Larson, and A. Hanjalic. Cross-domain collaborative filtering with factorization machines. In *ECIR*, pages 656–661. Springer, 2014. [Cited on page 125]

[126] J. Luo, X. Liu, Y. Zhang, D. Ye, and Z. Xu. Fuzzy trust recommendation based on collaborative filtering for mobile ad-hoc networks. In *LCN*, pages 305–311, 2008. [Cited on pages 17 and 60]

[127] Y. Luo, J. Le, and H. Chen. A privacy-preserving book recommendation model based on multi-agent. In *Computer Science and Engineering, 2009. WCSE'09. Second International Workshop on*, volume 2, pages 323–327. IEEE, 2009. [Cited on page 96]

[128] H. Ma. An experimental study on implicit social recommendation. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 73–82. ACM, 2013. [Cited on page 75]

[129] Mahout. http://mahout.apache.org. [Cited on page 31]

[130] Y. Mao, R. Morris, and M. F. Kaashoek. Optimizing mapreduce for multicore architectures. In *Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Tech. Rep.* Citeseer, 2010. [Cited on page 2]

[131] J. McAuley and J. Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 165–172. ACM, 2013. [Cited on page 116]

[132] F. McSherry and I. Mironov. Differentially private recommender systems: building privacy into the net. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 627–636. ACM, 2009. [Cited on pages 80, 88, 94, 96, and 97]

[133] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. Power management of online data-intensive services. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 319–330. IEEE, 2011. [Cited on pages 2 and 23]

[134] A. K.-B. Merialdo. Clustering for collaborative filtering applications. *Intelligent Image Processing, Data Analysis & Information Retrieval*, 3:199, 1999. [Cited on page 11]

[135] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. [Cited on page 67]

[136] B. N. Miller, J. A. Konstan, and J. Riedl. Pocketlens: Toward a personal recommender system. *ACM Transactions on Information Systems (TOIS)*, 22(3):437–476, 2004. [Cited on page 33]

[137] R. J. Mooney and L. Roy. Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 195–204. ACM, 2000. [Cited on page 89]

[138] *MovieLens*. http://grouplens.org/datasets/movielens/. [Cited on pages 28, 39, 51, 71, and 89]

[139] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 111–125. IEEE, 2008. [Cited on pages 2, 15, and 81]

[140] D. Novak and M. Batko. Metric index: An efficient and scalable solution for similarity search. In *Similarity Search and Applications, 2009. SISAP'09. Second International Workshop on*, pages 65–73. IEEE, 2009. [Cited on page 139]

[141] D. Oard and J. Kim. Implicit feedback for recommender systems. In *in Proceedings of the AAAI Workshop on Recommender Systems*, 1998. [Cited on page 75]

[142] V. C. Ostuni, T. Di Noia, E. Di Sciascio, and R. Mirizzi. Top-n recommendations from implicit feedback leveraging linked open data. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 85–92. ACM, 2013. [Cited on page 75]

[143] J. Pan and D. Manocha. Bi-level locality sensitive hashing for k-nearest neighbor computation. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 378–389. IEEE, 2012. [Cited on page 2]

[144] R. Patra, E. Samosvat, M. Roizner, and A. Mishchenko. Boostjet: Towards combining statistical aggregates with neural embeddings for recommendations. *arXiv preprint arXiv:1711.05828*, 2017. [Cited on page 6]

[145] R. Patra, E. Samosvat, M. Roizner, and A. Mishchenko. Boostjet: Towards combining statistical aggregates with neural embeddings for recommendations. *arXiv preprint arXiv:1711.05828*, 2017. [Cited on page 126]

[146] H. Polat and W. Du. Privacy-preserving collaborative filtering using randomized perturbation techniques. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 625–628. IEEE, 2003. [Cited on page 96]

[147] Python recsys. https://pypi.python.org/pypi/python-recsys/0.2. [Cited on page 72]

[148] Z. Qian, X. Chen, N. Kang, M. Chen, Y. Yu, T. Moscibroda, and Z. Zhang. Madlinq: large-scale distributed matrix computation for the cloud. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 197–210. ACM, 2012. [Cited on page 81]

[149] M. F. Rahman, W. Liu, S. Thirumuruganathan, N. Zhang, and G. Das. Privacy implications of database ranking. *Proceedings of the VLDB Endowment*, 8(10):1106–1117, 2015. [Cited on page 102]

[150] N. Ramakrishnan, B. J. Keller, B. J. Mirza, A. Y. Grama, and G. Karypis. Privacy risks in recommender systems. *IEEE Internet Computing*, 5(6):54, 2001. [Cited on pages 2, 6, 98, 102, 103, 106, and 110]

[151] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, pages 13–24. Ieee, 2007. [Cited on pages 2 and 31]

[152] A. M. Rashid, I. Albert, D. Cosley, S. K. Lam, S. M. McNee, J. A. Konstan, and J. Riedl. Getting to know you: learning new user preferences in recommender systems. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 127–134. ACM, 2002. [Cited on page 2]

[153] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 811–820. ACM, 2010. [Cited on pages 72 and 76]

[154] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *CSCW*, pages 175–186, 1994. [Cited on pages 11 and 65]

[155] D. Rosaci, G. M. Sarné, and S. Garruzzo. Muaddib: A distributed recommender system supporting device adaptivity. *ACM Transactions on Information Systems (TOIS)*, 27(4):24, 2009. [Cited on page 33]

[156] C. Sabottke, O. Suciu, and T. Dumitras. Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits. In *USENIX Security Symposium*, pages 1041–1056, 2015. [Cited on page 98]

[157] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295, 2001. [Cited on pages xvii, 4, 5, 11, 12, 40, 41, 44, 45, 59, 65, 87, 101, 104, 117, and 139]

[158] B. M. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Proceedings of the fifth international conference on computer and information technology*, volume 1, 2002. [Cited on page 39]

[159] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260. ACM, 2002. [Cited on page 3]

[160] G. Schröder, M. Thiele, and W. Lehner. Setting goals and choosing metrics for recommender system evaluations. In *CEUR Workshop Proc*, volume 811, pages 78–85, 2011. [Cited on page 17]

[161] Sequence-based recommendations. https://github.com/rdevooght/sequence-based-recommendations. [Cited on page 72]

[162] Seti@home. http://setiathome.berkeley.edu. [Cited on page 33]

[163] S. Shahrivari. Beyond batch processing: towards real-time and streaming big data. *Computers*, 3(4):117–129, 2014. [Cited on pages 2 and 41]

[164] B. Shapira, L. Rokach, and S. Freilikhman. Facebook single and cross domain data for recommendation systems. *User Modeling and User-Adapted Interaction*, pages 1–37, 2013. [Cited on page 102]

[165] U. Shardanand and P. Maes. Social information filtering: algorithms for automating “word of mouth”. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217. ACM Press/Addison-Wesley Publishing Co., 1995. [Cited on page 117]

[166] J. Shi, Y. Qiu, U. F. Minhas, L. Jiao, C. Wang, B. Reinwald, and F. Özcan. Clash of the titans: Mapreduce vs. spark for large scale data analytics. *Proceedings of the VLDB Endowment*, 8(13):2110–2121, 2015. [Cited on page 37]

[167] Y. Shi, M. Larson, and A. Hanjalic. Tags as bridges between domains: Improving recommendation with tag-induced cross-domain collaborative filtering. *User Modeling, Adaption and Personalization*, pages 305–316, 2011. [Cited on page 124]

[168] Shopzilla, inc. privacy policy. http://about.bizrate.com/privacy-policy. [Cited on page 2]

[169] P. Skočir, I. Bojić, and G. Ježić. Implementation of agent-based games recommendation system on mobile platforms. In *Agent and Multi-Agent Systems: Technologies and Applications*, pages 67–76. Springer, 2014. [Cited on page 72]

[170] I. Soboroff and C. Nicholas. Collaborative filtering and the generalized vector space model. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 351–353. ACM, 2000. [Cited on page 11]

[171] F. Soldo, A. Le, and A. Markopoulou. Predictive blacklisting as an implicit recommendation system. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010. [Cited on pages 60 and 75]

[172] Apache spark. https://spark.apache.org/. [Cited on pages 68 and 70]

[173] E. I. Sparling and S. Sen. Rating: how difficult is it? In *Proceedings of the fifth ACM conference on Recommender systems*, pages 149–156. ACM, 2011. [Cited on page 74]

[174] X. Su and T. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009. [Cited on page 4]

[175] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4, 2009. [Cited on pages 23, 83, and 86]

[176] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment*, 4(11):992–1003, 2011. [Cited on page 106]

[177] M. N. Szomszor, I. Cantador, and H. Alani. Correlating user profiles from multiple folksonomies. In *Proceedings of the nineteenth ACM conference on Hypertext and hypermedia*, pages 33–42. ACM, 2008. [Cited on page 124]

[178] C. Taranto, N. Di Mauro, and F. Esposito. Learning in probabilistic graphs exploiting language-constrained patterns. In *International Workshop on New Frontiers in Mining Complex Patterns*, pages 155–169. Springer, 2012. [Cited on page 72]

[179] M. M. Tuffield, A. Loizou, and D. Dupplaw. The semantic logger: Supporting service building from personal context. In *Proceedings of the 3rd ACM workshop on Continuous archival and retrival of personal experences*, pages 55–64. ACM, 2006. [Cited on page 124]

[180] L. H. Ungar and D. P. Foster. Clustering methods for collaborative filtering. In *AAAI workshop on recommendation systems*, volume 1, pages 114–129, 1998. [Cited on page 11]

[181] M. Van Dijk and A. Juels. On the impossibility of cryptography alone for privacy-preserving cloud computing. *HotSec*, 10:1–8, 2010. [Cited on pages 15 and 81]

[182] R. Van Meteren and M. Van Someren. Using content-based filtering for recommendation. In *Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop*, pages 47–56, 2000. [Cited on page 97]

[183] S. Voulgaris and M. Van Steen. Epidemic-style management of semantic overlays for content-based searching. In *European Conference on Parallel Processing*, pages 1143–1152. Springer, 2005. [Cited on pages 24, 25, and 26]

[184] K. Whitenton. Minimize cognitive load to maximize usability. *Erişim Tarihi*, 20:2016, 2013. [Cited on page 75]

[185] *Wiki-Elections dataset*. https://snap.stanford.edu/data/wiki-Elec.html. [Cited on pages 50 and 51]

[186] X-Map GitHub repository. https://github.com/LPD-EPFL-ML/X-MAP. [Cited on pages 112 and 113]

[187] X. Yang, Z. Zhang, and K. Wang. Scalable collaborative filtering using incremental update and local link prediction. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2371–2374. ACM, 2012. [Cited on pages 4 and 59]

[188] K. Yu, S. Zhu, J. Lafferty, and Y. Gong. Fast nonparametric matrix factorization for large-scale collaborative filtering. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 211–218. ACM, 2009. [Cited on page 59]

[189] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory

cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012. [Cited on pages 103 and 116]

[190] N. Zeilemaker, M. Capotă, A. Bakker, and J. Pouwelse. Tribler: P2p media search and sharing. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 739–742. ACM, 2011. [Cited on page 33]

[191] F. Zhang, T. Gong, V. E. Lee, G. Zhao, C. Rong, and G. Qu. Fast algorithms to evaluate collaborative filtering recommender systems. *Knowledge-Based Systems*, 96:96–103, 2016. [Cited on page 72]

[192] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing. Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters. In *USENIX ATC*, pages 181–193, 2017. [Cited on page 37]

[193] J. Zhang, C. Wang, J. Wang, and J. X. Yu. Inferring continuous dynamic social influence and personal preference for temporal behavior prediction. *Proceedings of the VLDB Endowment*, 8(3):269–280, 2014. [Cited on page 2]

[194] S. Zhang, J. Ford, and F. Makedon. Deriving private information from randomly perturbed ratings. In *Proceedings of the 2006 SIAM International Conference on Data Mining*, pages 59–69. SIAM, 2006. [Cited on page 96]

[195] S. Zhang, G. Wu, G. Chen, and L. Xu. On building and updating distributed lsi for p2p systems. In *Parallel and Distributed Processing and Applications-ISPA 2005 Workshops*, pages 9–16. Springer, 2005. [Cited on page 33]

[196] W. Zhang, S. Gupta, X. Lian, and J. Liu. Staleness-aware async-sgd for distributed deep learning. In *IJCAI*, pages 2350–2356, 2016. [Cited on pages 4 and 35]

[197] Q. Zhao, W. Zuo, Z. Tian, X. Wang, and Y. Wang. Predicting trust relationships in social networks based on wknn. *JSW*, 10(1):71–81, 2015. [Cited on pages 4, 17, and 60]

[198] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the netflix prize. *Lecture Notes in Computer Science*, 5034:337–348, 2008. [Cited on page 59]

[199] T. Zhu, G. Li, Y. Ren, W. Zhou, and P. Xiong. Differential privacy for neighborhood-based collaborative filtering. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 752–759. ACM, 2013. [Cited on pages 103, 111, 112, and 117]

[200] T. Zhu, Y. Ren, W. Zhou, J. Rong, and P. Xiong. An effective privacy preserving algorithm for neighborhood-based collaborative filtering. *Future Generation Computer Systems*, 36:142–155, 2014. [Cited on pages 103, 111, 112, and 117]

Curriculum Vitae

Rhicheek Patra
Route Cantonale 37, Saint-Sulpice, Switzerland
• rhicheek.patra@epfl.ch • +41 779446054

EDUCATION

École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland

- Ph.D. in Computer Science — Sep 2013 – Jan 2018
  - Adviser: Prof. Rachid Guerraoui
  - Focus: Distributed Machine Learning, Recommender Systems, Privacy, Online Social Networks, Big Data Analytics.

Indian Institute of Technology (I.I.T), Kharagpur, West Bengal, India

- B.Tech. in Computer Science and Engineering — Jun 2009 – May 2013
  - Thesis: *SceMat* - An Efficient Scene Matcher for Images.

RESEARCH EXPERIENCE

Oracle Labs, Zurich, Switzerland

- Research Intern — Jun 2017 – Sep 2017
  - Graph-Empowered Machine Learning.

Yandex, Moscow, Russia

- Research Intern — Jun 2016 – Aug 2016
  - MXJet: An ensemble of several recommenders, leveraging Yandex's proprietary boosted decision tree (Matrixnet), implemented in production infrastructure to provide real-time personalized advertisements to tens of millions of users.

Technicolor, Rennes, France

- Research Intern — Jun 2015 – Aug 2015
  - *iKNN:* An implicit recommender leveraging consumption order of users.
  - *I2:* An implicit item-based recommender using time-based partitioning.
  - *G2G:* A novel method to detect any change in user preferences leveraging temporal trajectories.

NetApp, Bangalore, India

- Research Intern — May 2012 – Jul 2012
  - Design and implementation of a dynamic infrastructure to control the backup load (dump) in a system, in order, to allow maximum resource utilization when multiple backup sessions are active.
  - Implementation of a global message pool to speed up the allocation of messages in file-dump phase and thereby improve the speed of dump.

Indian Statistical Institute, Kolkata, India

- Research Intern — May 2011 – Jul 2011
  - Developed an approximation algorithm for Guard Placement Problem to compute the minimal number of guards to connect the clusters scattered on an x-monotone terrain.

PUBLICATIONS

[6] R. Guerraoui, A.-M. Kermarrec, R. Patra, M. Valiyev, and J. Wang (*alphabetical order*),
"I know nothing about you but here is what you might like",
DSN, Denver, Colorado, USA, 2017.

[5] R. Guerraoui, A-M. Kermarrec, T. Lin and R. Patra (*alphabetical order*),
"Heterogeneous Recommendations: What You Might Like To Read After Watching Interstellar",
PVLDB, Volume 10, No. 10, Munich, Germany, 2017.

[4] G. Damaskinos, R. Guerraoui and R. Patra (*alphabetical order*),
"Capturing the Moment: Lightweight Similarity Computations",
ICDE, San Deigo, CA, USA, 2017.

[3] R. Guerraoui, E. Merrer, R. Patra and B.D. Tran (*alphabetical order*),
"Frugal Topology Construction for Stream Aggregation in the Cloud",
INFOCOM, San Francisco, CA, USA, 2016.

[2] R. Guerraoui, A.-M. Kermarrec, R. Patra, and M. Taziki (*alphabetical order*),
"D2P: Distance-Based Differential Privacy in Recommenders",
PVLDB, Volume 8, No. 8, Hawaii, USA, 2015.

[1] A. Boutet, D. Frey, R. Guerraoui, A.-M. Kermarrec, and R. Patra (*alphabetical order*),
“Hyrec: Leveraging browsers for scalable recommenders”,
MIDDLEWARE, Bordeaux, France, 2014.

PATENTS

[1] E. Le Merrer, R. Patra, and J.-R. Vigouroux (*alphabetical order*),
“Device and method for finding top-k neighbours in a recommendation system”,
EP Patent, EP3142059 A1, 2017.

ONGOING WORK

[3] R. Guerraoui, E. Merrer, R. Patra and J.-R. Vigouroux (*alphabetical order*),
“Sequences, Items And Latent Links: Recommendation With Consumed Item Packs”.

[2] R. Patra, E. Samosvat, M. Roizner and A. Mishchenko,
“MX-Jet: Towards Combining Statistical Aggregates with Neural Embeddings for Recommendations”.

[1] G. Damaskinos, R. Guerraoui and R. Patra (*alphabetical order*),
“Mobile Learning: Distributed Machine Learning on Mobile Devices”.

EXPERIENCE

Programming Languages

- *Expert:* Java, Python
  Others: C, HTML, CSS, PHP, LaTeX

Distributed (Big Data) Platforms

- Spark, Tensorflow, Hadoop, Yarn

Integrated Development Environments

- Eclipse, NetBeans, Android Studio

Operating Systems

- GNU/Linux, MacOS, Windows

Extra-curricular Activities

- Quiz, National Service Scheme (NSS), National Science Olympiad (NSO)

OTHER PROJECTS & SOFTWARES

Understanding Viral Popularity, I.I.T-Kharagpur Jan 2013 – Apr 2013

- This project aims at understanding the popularity of viral videos like Gangnam Style through extensive data and social network analysis.

TwitMiner, I.I.Sc-Bangalore Jan 2013 – Apr 2013

- This software provides a python implementation for a Naive Bayes classifier, integrated with Adaboost, to classify tweets. It ranked 8^{th} in the leaderboard with a prediction accuracy of 94.891% on the test set.

TripSelector, I.I.T-Kharagpur Jul 2012 – Nov 2012

- This software implements a trip planner, in Prolog, to plan trips between various source, destination pairs based on the shortest distance as well as the intermediate nodes to be covered.

FileSys: A FileSystem, I.I.T-Kharagpur Jan 2012 – Apr 2012

- FileSys is a complete file system implemented in C.

MemSys: A Memory Management System, I.I.T-Kharagpur Jan 2012 – Apr 2012

- MemSys is a complete memory management system implemented in C.

C-Comp: A customized compiler, I.I.T-Kharagpur Jul 2011 – Nov 2011

- This compiler is designed for a customized C-type language using Flex, Bison, Yacc.

SC-RISC: Customized RISC Processor, I.I.T-Kharagpur Jan 2011 – Apr 2011

- SC-RISC is a single-cycle RISC processor implemented in Verilog.

Automated Courier System, I.I.T-Kharagpur Jul 2010 – Nov 2010

- This is a computerized software, implemented in Java, for a courier company to manage all the tasks of the company through computers.

TEACHING & MENTORSHIP

Teaching

- Distributed Algorithms (CS-451), Graduate class, EPFL, 2015-2016
- Programming (CS-111, CS-112), Undergraduate class, EPFL, 2014-2017

Mentorship

- Mihaela Turcu, EPFL — Feb 2014 – May 2014
  “P3: Persistent-client based Privacy Preservation in Recommenders”,
- Mahammad Valiyev, TU Munich — Jul 2014 – Sep 2014
  “Towards a Confidential Recommender System”
- Stéphane Henriot, ENS Paris — Oct 2014 – Jul 2015
  “Recommendation Systems and Privacy”
- Tao Lin, EPFL — Jan 2015 – Jan 2016
  “Private cross-domain recommender”
- Georgios Damaskinos, EPFL — Sep 2015 – Jan 2016
  “Energy-efficient recommenders”
- Mengjie Zhao, EPFL — Jan 2016 – May 2016
  “DeMF: an Online Recommender based on Matrix Factorisation”
- Wei Ma, EPFL — Jan 2016 – May 2016
  “Distributed recommendation computation on client machines”
- Guillaume Aubian, ENS Cachan — May 2016 – Sep 2016
  “Towards Distributed Machine Learning with Android Devices”
- Jinbu Liu, EPFL — Sep 2016 – Jan 2017
  “Democratizing Online Machine Learning”
- Manana Lortkipanidze, EPFL — Sep 2016 – Jan 2017
  “Distributed Mini-Batch SGD with crash-stop and crash-recovery”
- Safae Lhazmir, Université Mohammed V — Nov 2016 – Jan 2017
  “Machine Learning for predicting Facebook Check-ins”
- Tasho kjosev, EPFL — Jan 2017 – May 2017
  “Decentralized P2P Machine Learning”
- Hakan Gökcesu, EPFL — Jan 2017 – May 2017
  “Fault-tolerant Machine Learing using Importance Sampling”
- Alexis Cartier & Andreas Yazdani, EPFL — Jan 2017 – May 2017
  “Byantine-tolerant Asynchronous Machine Learning”
- Jinbu Liu, EPFL — Oct 2017 – Jan 2018
  “Compressing deep neural networks for mobile devices”

LANGUAGES

- *Bengali*: Native language.
- *English*: Fluent (reading); fluent (speaking, writing).
- *Hindi*: Intermediate (reading); basic (speaking, writing).

AWARDS

- President’s Gold Medal
  For being the topper in the State Board Exam.
- Madhyamik Topper Award,
  Madhyamik Topper Award, provided by the Indian Citizen Organisation, for academic excellence in the Board Examination.
- Top-10 in TwitMiner Competition
  8^{th} position all over India in the TwitMiner Competition, held by IISC Bangalore, for classifying tweets with 95% accuracy.