PhD-FSTC-2018-61
The Faculty of Sciences, Technology and Communication

DISSERTATION

Defence held on 19/10/2018 in Esch-sur-Alzette
to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Jun WANG
Born on 25th January 1986 in Sichuan, China

PRIVACY-PRESERVING RECOMMENDER SYSTEMS FACILITATED BY THE MACHINE LEARNING APPROACH

Dissertation defence committee

Thesis advisor: Peter Y.A. RYAN Jun WANG

Privacy-preserving Recommender Systems Facilitated By The Machine Learning Approach

Abstract

Recommender systems, which play a critical role in e-business services, are closely linked to our daily life. For example, companies such as Youtube and Amazon are always trying to secure their profit by estimating personalized user preferences and recommending the most relevant items (e.g., products, news, etc.) to each user from a large number of candidates. State-of-the-art recommender systems are often built on-top of collaborative filtering techniques, of which the accuracy performance relies on precisely modeling user-item interactions by analyzing massive user historical data, such as browsing history, purchasing records, locations and so on. Generally, more data can lead to more accurate estimations and more commercial strategies, as such, service providers have incentives to collect and use more user data. On the one hand, recommender systems bring more income to service providers and more convenience to users; on the other hand, the user data can be abused, arising immediate privacy risks to the public. Therefore, how to preserve privacy while enjoying recommendation services becomes an increasingly important topic to both the research community and commercial practitioners.

The privacy concerns can be disparate when constructing recommender systems or providing recommendation services under different scenarios. One scenario is that, a service provider wishes to protect its data privacy from the inference attack, a technique aims to infer more information (e.g., whether a record is in or not) about a database, by analyzing statistical outputs; the other scenario is that, multiple users agree to jointly perform a recommendation task, but none of them is willing to share their private data with any other users. Security primitives, such as homomorphic encryption, secure multiparty computation, and differential privacy, are immediate candidates to address the privacy concerns. A typical approach to build efficient and accurate privacy-preserving solutions is to improve the security primitives, and then apply them to existing recommendation algorithms. However, this approach often yields a solution far from the satisfactory-of-practice, as most users have a low tolerance to the latency-increase or accuracy-drop, regarding recommendation services.

The PhD program explores machine learning aided approaches to build efficient privacy-preserving solutions for recommender systems. The results of each proposed solution demonstrate that machine learning can be a strong assistant for privacy-preserving, rather than only a troublemaker.

Contents

Listing of figures

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisor Prof. Dr. Peter Y.A. Ryan for his continuous support during my doctoral study, for his patience and immense knowledge. He has also offered a maximum freedom, so that I can conduct research and explorations in topics I am interested in.

My sincere gratitude also goes to my co-supervisor Dr. Qiang Tang, for introducing me to the area of privacy-preserving. His guidance and encouragement helped me in all the time of research and writing of this dissertation.

I am grateful for the jury members for their interest in this dissertation and taking their valuable time to evaluate this dissertation.

Finally, I could not have gone so far without my beloved wife (Fengyun Ju) and my parents (Shunyou Wang and Chunhua Chen) in my back. They are the sunshine in my life and their support has always been warm and unreserved. I cannot imagine surviving this journey without their support.

1
Introduction

Recommender system is one of the most frequently used machine learning applications, which is closely linked to our daily life. For example, a user can always receive personalized item recommendations (e.g., products, videos, etc.) when visiting websites like Amazon and Youtube. This is because their recommender systems have estimated the user preferences. Collaborative filtering, such as matrix factorization [64] and neighborhood-based method [25], is the dominant technique to construct personalized recommender systems [25, 114]. Massive user historical data, such as browsing records and locations, is indispensable to accurately model user-item interactions, e.g., user-product interactions, user-location interactions, etc. Generally, the data from a large number of users allows recommender systems to precisely extract the commonality of all the users and items, thus to estimate user preferences on non-observed items; more data of every single user enables the system more precisely capturing each user's difference from others, thus to provide more accurate personalized recommendation services. In addition to this, the user preference is often influenced by social information such as advertisements and friends' opinion, and also can be shifted with time [64, 114]. In order to maintain the recommendation accuracy, the recommender systems have to keep collecting new user activities. All of the facts make the systems aggressive on all kinds of user data.

With recommender systems, users can get preferred products from a vast number of candidates efficiently. At the same time, the user data is also exposed to the service provider or a third party. This may result in immediate privacy risks to the users, especially, when the data contains sensitive information such as locations and purchase records. How to protect the data privacy while still enjoying accurate recommendation services, it has been an important topic to both the research community and practitioners.

1.1 Privacy Issues in Recommender Systems

Let's go back to 1987, the year of Robert Bork's confirmation hearings for the U.S. Supreme Court. A video store leaked Bork's movie rental records to a reporter for the Washington City Paper. Then the paper published the records likely in an attempt to embarrass Bork [127]. This event directly led to the enactment of the 1988 Video Privacy Protection Act, which was created to prevent what it refers to as "wrongful disclosure of video tape rental or sale records (or similar audio visual materials, to cover items such as video games and the future DVD format)" [35].

Removing identifying information, such as social security number, name, and address, does not guarantee privacy, since exploiting the remaining information, e.g., zip code, gender, etc., can still identify a person uniquely [115]. Anonymity techniques, such as k-anonymization [7, 115] and t-closeness [70] have been proposed to prevent de-anonymization from quasi-identifiers. However, these techniques suffer from inference attacks when auxiliary information is available. In fact, releasing data with these techniques is susceptible to attacks which carry some auxiliary information. Machanavajjhala et al. [74] showed that leveraging the association between one or more quasi-identifier attributes with a sensitive attribute can reduce the set of possible values for this sensitive attribute. As an instance, they use the knowledge of heart attacks occurred at a reduced rate in Japanese patients to narrow the range of values for a sensitive attribute of a patient's disease. Narayanan et al. [82] presented a class of robust statistical de-anonymization attacks against high-dimensional micro-data, such as individual preferences, recommendations, transaction records and so on, under an assumption that attackers have some background knowledge.

Leading e-commercial companies sell products covering not only movies and books, but also various categories of products such as adult toys, health devices, costume, daily supplies, and also on. In order to construct more accurate and personalized recommender systems, e-commercial companies have to collect various kinds of user data, such as demographic information (e.g., age, gender, occupation, etc.) and user historical information (e.g., purchasing, browsing, location, etc.) [3]. With these data, they can portray users precisely, inferring their personal information such as gender, age, consumption habits, health status. For example, the retail company Target showed that they could accurately predict the pregnancy of their female customers, recommending personalized coupons to them [31]. Apparently, these kind of techniques can be easily extended to infer the health status or financial status of a user. On the one hand, users may be unwilling to disclose such kind of information; on the other hand, user data may be abused such as selling the data to a third party which may lead a loss to users. Therefore, recommender systems can bring immediate risks to the users, due to a privacy breach.

Providing recommendation services without considering privacy-preserving, it as well brings risks to these e-commercial companies. Since user data can be monetized, and competitors can also analyze user preferences and behaviors to adjust their business tactics. Deploying a physical security layer, such as firewall, can prevent direct data leakage. However, attackers can still infer user privacy via recommendation results released by a service provider [82]. Users may also give up this service provider when they realize that their privacy is in risks.

1.2 Related Work and Challenges

Privacy-preserving machine learning (including recommender systems) can be generally described as a problem that data owners wish to obtain machine learning services without directly or indirectly revealing their private data. With this fact, this dissertation classifies the privacy-preserving solutions for recommender systems into the following two categories:

- Centralized case. In this case, a trusted server, which can fully access the user data, wishes to provide recommendations to users in its database, while protecting user (data) privacy from the inference attack. The inference attack refers to an attack technique which illegitimately gains knowledge about a record or database by analyzing statistical outputs.

- Distributed case. In this case, multiple data owners collaboratively perform a recommendation task, but none of them is willing to disclose their private data. Normally, the privacy-preserving solutions are constructed without assuming a (fully) trusted server.

1.2.1 Centralized Case

[**Case Description**] *A trusted server provides recommendation services to users whose data that it can fully access. The users obtain the recommendation results, and their capability of inferring others' data privacy is strictly quantified (i.e., the privacy loss of a user is quantifiable and controllable).*

Accurately answering queries and privacy-protection are always two competing goals at stake, there is no free lunch in terms of accuracy and privacy [26, 59]. Shokri et al. [110] have demonstrated that more information, such as whether a record is in in a database or not, can be inferred by exploiting prediction or classification outputs (i.e., via inference attack). Differential privacy [33] is the prevalent approach to address such kind of risks. It strictly quantifies and controls the privacy loss when responding to a query with an asymptotically true answer. Intuitively, differential privacy provides a participant a possibility to deny that it participated in a computation, by introducing uncertainties. According to the places where existing differentially private solutions introduced the uncertainties, we can classify these solutions into five categories: input perturbation, output perturbation, objective function perturbation, gradient perturbation and sampling.

- Input perturbation. The server first adds calibrated noise into training data directly; and then it can perform any machine learning tasks without compromising the privacy guaranteed by the noise, according to the post-processing theorem [76]. Arnaud et al. [9] stated that this simple method can preserve the accuracy of matrix factorization with a modest privacy budget.

- Output perturbation. The server adds noise into the outputs of a machine learning model or components decomposed from a model. McSherry et al. [76] applied differential privacy to recommendation algorithms which don't require a dynamic training process, such as neighborhood-based methods and global effects. They first factor the algorithms into different components, of which the sensitivity [32] can be easily estimated; then they add noise to the output of each component, based on their sensitivity. Chaudhuri et al. [21] and Arnaud et al. [9] examined the privacy-accuracy tradeoff of output perturbation regarding logistic regression and matrix factorization, respectively. The two algorithms need a dynamic training process such as stochastic gradient descent.

- Objective function perturbation. The server bounds the privacy loss by customizing a noise term into the objective function of a machine learning model. Chaudhuri et al. [21] and Zhang et al. [131] analyzed how to deploy this approach to logistic regression. Arnaud et al. [52] and Phan et al. [95] extended their solutions to build differentially private matrix factorization and deep autor-encoder [85], respectively.

- Gradient perturbation. The server often guarantees the privacy by adding noise into the error information used to calculate gradients. Song et al. [112] presented a general approach to differential-privately update gradients in each training iteration. Arnaud et al. [9] and Martin et al. [2] analyzed how to use the gradient perturbation method to build differential privacy into matrix factorization and deep learning models, respectively. In their solutions, some common tricks, such as clipping error information and reducing the number of training iterations, are adopted to achieve a better trade-off between privacy and accuracy.

- Sampling. There are two sub-branches, one is randomly discarding or partitioning data points used to learn responses, thus to reduce privacy leakage [20, 39, 87]. With this kind of methods, Zhu et al [134] suggested randomly selecting neighbors to preserve neighborhood information; Martin et al. [2] naturally reduced the privacy leakage by a mini-batch stochastic optimization method [69]. The other is sampling from scaled posterior distribution [124]. Liu et al. [73] applied this method to building differentially private matrix factorization.

1.2.1.1 Challenges

Various methods for applying differential privacy into machine learning tasks have been practiced in the past years. However, they are still far from reaching the point that both the privacy and accuracy are sufficiently preserved. The input perturbation method was stated that it was superior than the gradient perturbation and output perturbation approaches, when building differential privacy into matrix factorization [9]. But it requires a further evaluation when applying to different machine learning tasks or training matrix factorization on a larger database, since the method presented in [9] did not take into consideration (at least) the fact that gradient perturbation methods allow incorporating sampling methods to amplify privacy. Moreover, poisoned training data may also bias the ground truth, squeezing the space of accuracy improvements. The output perturbation method is often applied to building differential privacy into models which do not need a iterative training process, such as neighborhood-based methods using Cosine similarity. However, using this method to differential-privately learn models, such as matrix factorization and deep learning, it often leads to a significant accuracy loss since the sensitivity bound could be multiplicative-ly amplified by the iterative training process. The gradient perturbation method is a commonly adopted approach for models learned iteratively. According to the composition theory of differential privacy [33], sequentially applying differential privacy to a database leads to a linear increase of privacy loss. Usually, these iterative training methods require dozens to thousands of iterations for convergences (i.e., to get optimal model parameters). The objective function perturbation method can learn a model differential-privately, where the privacy loss is independent of the number of training iterations. However, when a model contains multiple parameter spaces, it is challenging to analyze the privacy guarantee as the parameter spaces can impact mutually. As to the sampling method, a recent observation shows that releasing a sample from the scaled posterior distribution of a Bayesian model can be sufficiently differential privacy [124]. It allows differential-privately sampling a model with multiple parameter spaces, of which the privacy guarantee is also independent of the number of training iterations. This approach has two assumptions, the first is that there must be a Bayesian model and the log-likelihood of which is bounded; the second is that the difference (δ) between the distribution where the sample is from and the true distribution is asymptotically close to zero (i.e., $\delta \rightarrow 0$). The difference δ can compromise the privacy at a speed of $\mathcal{O}(e^{\delta})$, which is non-trivial. Though Teh et al. [121] proved that the convergence (i.e., $\delta \rightarrow 0$) can be guaranteed in a large (but finite) number of sampling iterations, there are two issues cannot be simply ignored, one is that any flaws of engineering implementation would lead to an un-controllable privacy loss; the other is that Markov Chain Monte Carlo (MCMC) sampling process is extremely inefficient.

1.2.2 Distributed Case

[**Case Description**] *There are multiple users (data owners), each of them has a piece of data $\mathcal{D}_i$. They jointly perform a recommendation task on the data set $\{\mathcal{D}_i\}_i^n$. Ideally, each user learns only the output (i.e., recommendation results or a recommendation* *model) without leaking any of their private data, as if there is a trusted server. In the real word, we often assume a semi-honest threat model, where the users always follow the secure protocol that they agreed on but curious with others' data. The users are also allowed to obtain what can be inferred from the outputs.*

Cryptographic primitives, such as homomorphic encryption [40], secure multiparty computation [71] and secret sharing [94], are commonly adopted tools to prevent private data from being accessed by attackers. Specific privacy-preserving scenarios/solutions often depend on the data and computational resource distribution. Based on this fact, we can divide existing solutions into two categories: Each party jointly and equally participates into a machine learning task, when they have a similar amount of data and sufficient computational resource; data owners outsource computations to a third party, such as a cloud service provider which has a powerful computation infrastructure and broad machine learning knowledge.

For the first category, a machine learning task is often decomposed into a sequence of vector-additive operations, turning the privacy-preserving demand to a problem of secure private aggregations. Canny et al. [17, 18] focused on privately performing SVD (Singular-value decomposition), where they compute aggregations with resort to homomorphic encryption [40] and secret sharing [94] techniques. In their solution, they don't require a central server but assume that at least half of the computation participants are not corrupted. Duan et al. [30] assumed that there are a few of servers, where at least one server is uncorrupted. The data owners (where at least half of them are uncorrupted) secretly share their data on some of these servers (at least two), where each server learns only random numbers if without having all the other shares. One of the servers is selected to perform aggregations (the results of which are often assumed to be public) and returns the global computation results. Bonawitz et al. [12] proposed a solution in which each user secretly shares their data to all the other users, a semi-trusted global server is put to aggregate the users' intermediate data. A primary contribution of their work is to efficiently and securely address the problem of user dropout. Shokri et al. [108] proposed a solution for multiple data owners to jointly train a deep learning model, where they assumed a semi-trusted server allowed to access intermediate values (gradients). Though they stressed that the private data is not directly exposed to any others, the gradients carry the error information (i.e., the difference between the predictions and the ground-truth) computed from the data of each party, providing rich information to a potential attack. Note that in their assumption, the semi-trusted server learns not only the aggregations, but also the exact inputs of each party.

For the second category, users send their encrypted data to the cloud, where the privacy-preserving task becomes a problem of how to efficiently perform machine learning tasks on encrypted data. Nikolaenko et al. [86] presented a privacy-preserving solution for matrix factorization with an assumption of two non-colluding servers, where one server serves as the cryptography service provider (CSP), the other (RecSys) evaluates recommendation algorithms (e.g., matrix factorization). Each user submits encrypted item-rating pairs to the RecSys. The RecSys masks these pairs and forwards them to the CSP. The CSP decrypts and embeds them in a garbled circuit which is then sent to the RecSys for evaluation. The garbled values of the masks are obtained by the RecSys through oblivious transfer [100]. Nayak et al. [83] aim to build a secure computation framework (based on garbled circuits) that can easily use parallelization programming paradigms, where they assume two non-colluding cloud providers and both of them have parallel computing architectures. Mohassel et al. [80] further improved the efficiency performance of the secure framework of two non-colluding servers, by using secret sharing. Ohrimenko et al. [89], Hunt et al. [54] and Hynes et al. [56] built general privacy-preserving solutions based on trusted processors (e.g., Intel SGX-processors [75]), where they assume attackers cannot manipulate the content in the trusted processors.

Privacy-preserving Machine Learning as a Service (MLaaS) emerges as MLaaS has recently become a popular commercial paradigm. Privacy-preserving MLaaS is in fact a special case of the second category. It is often a two-party computation scenario where one party (client) holds data and wishes to get machine learning services, while the other party (cloud) provides the required service. Usually, they assume that the cloud has already trained a machine learning model, and obliviously execute the machine learning task by taking as input the the client's data.

Raphael et al. [14] designed a generic MLaaS privacy-preserving framework for a family of classifiers such as Naive Bayes and decision trees, based on a set of homomorphic encryption schemes. The primary contribution is a bunch of interactive protocols for securely computing non-linearities (e.g., comparison) which are not compatible with homomorphic encryption schemes. Gilad-Bachrach et al. [43] proposed a privacy-preserving solution for evaluating neural networks on encrypted data, based on a leveled homomorphic encryption scheme [15]. To be compatible with the homomorphic encryption scheme, they substitute state-of-the-art non-linear activation function (e.g., $ReLu(x) = max(0, x)$) with polynomials. Liu et al. [72] combined homomorphic encryption and garbled circuits to evaluate neural networks, that non-linearities can be correctly computed. Recently, Hunt et al. [55] introduced using trusted processors to construct privacy-preserving solutions for MLaaS, which also allows securely performing a model training phase on the cloud.

1.2.2.1 Challenges

State-of-the-art machine learning algorithms often rely on complex model structures (e.g., a multiple-layer structure with non-linear transformations) and massive training data (leading to a large volume of computations). Unfortunately, mapping messages from a plain-text space into a cipher space often results in a significant increase of the spatial and computational complexity and a confinement on the types and number of algebraic operations. For example, homomorphic encryption schemes support only additions and multiplications. In particular, partial homomorphic encryption schemes only allow either addition operations (e.g., Paillier cryptosystem [91]) or multiplication operations (e.g., ElGamal encryption scheme [34]), while cannot satisfy both. Somewhat (fully) homomorphic encryption schemes, such as[15, 37, 41], allow performing a limited number of addition and multiplication operations but pay the cost of efficiency. Secure multiparty computation [71] and garble circuits [130] support various algebraic operation in a cipher space. However, this kind of methods require computation participants to be online constantly, introducing an additional communication cost. Privacy-preserving solutions based on these cryptographic primitives often struggle in the trade-off of utility and privacy, where utility can be accuracy, efficiency and user experience. Using trusted hardware, such as Intel SGX [75], to build efficient privacy-preserving solutions recently has attracted a lot of attentions [55, 89]. This approach also suffers from a number of security limitations, such as SGX page faults, cache timing, processor monitoring and so on [54].

1.3 Our Contributions

The research work during the course of PhD program falls into both of the two categories, centralized case and distributed case, which are summarized as follows,

1. (*Centralized case.*) We apply the differential privacy concept to neighborhood-based recommendation methods (NBMs) under a probabilistic framework. We first present a solution, by directly calibrating Laplace noise into the training process, to differential-privately find the maximum a posteriori parameters similarity. Then we connect differential privacy to NBMs by exploiting a recent observation that sampling from the scaled posterior distribution of a Bayesian model results in provably differentially private systems. Our experiments show that both solutions allow promising accuracy with a modest privacy budget, and the second solution yields better accuracy if the sampling asymptotically converges. We also compare our solutions to the recent differentially private matrix factorization (MF) recommender systems, and show that our solutions achieve better accuracy when the privacy budget is reasonably small. This is an interesting result because MF systems often offer better accuracy when differential privacy is not applied.

2. (*Distributed case – A collaborative task.*) Recommender systems provide recommendations by exploiting the association between individual and popularity. Therefore, it requires massive training data, which in turn demands a

large number of computations. This fact always leads to a efficiency bottleneck when using cryptographic primitives to overcome security concerns. We use social-context to reduce computational cost while preserving the accuracy performance. Consider the following illustrative example, a user would like to know more about a product (e.g., a movie or a mobile phone), she is more likely to ask opinions from her friends who consumed the product. Therefore, a small number of influential friends may play a critical role in a person's decision. On the one hand, exploiting friends information we can significantly reduce the computational complexity. On the other hand, the privacy protection among friend is even more serious, as friends know more background information and have more impact to each other. We first demonstrate that friends have more common interests than ordinary. Then we show that using the data from only the friends and a few of strangers can result in promising accuracy performance. Lastly, based on our observations, we build secure recommendation protocols with resort to homomorphic encryption schemes. Thanks to the significant reduction of computational complexity, our solution allows efficiently computing recommendations on encrypted data.

3. (*Distributed case – Machine Learning as a Service.*)State-of-the-art recommender systems often rely on non-linear operations, or require training the recommendation model with the Client's data. While improving cryptographic tools (e.g., HE or SMC) is one typical way to improve the efficiency of privacy-preserving solutions, unfortunately, the improvement is usually far from satisfactory to make these solutions practical enough. We tackle this problem from the direction of designing crypto-friendly machine learning algorithms, so that we can achieve efficient solutions by directly using existing cryptographic tools. In particular, we propose CryptoRec, a new non-interactive secure 2PC protocol for RaaS, the key technical innovation of which is an HE-friendly recommender system. This recommendation model possesses two important properties: (1) It uses only addition and

multiplication operations, so that it is straightforwardly compatible with HE schemes. With this property, CryptoRec is able to complete recommendation computations without requiring the Server and the Client to be online continuously. Simply put, the Client sends her encrypted rating vector to the Server, then the Server computes recommendations with the Client's input and returns the results in an encrypted form. In addition to this, there is no other interaction between the two parties; (2) It can automatically extract personalized user representations by aggregating pre-learned item features, that we say the model has an item-only latent feature space. This property allows the Server with a pre-trained model to provide recommendation services without a tedious re-training process, which significantly improves the efficiency performance. Note that the Client's data is not in the Server's database which is used for model training. We demonstrate the efficiency and accuracy of CryptoRec on three real-world datasets. CryptoRec allows a server with thousands of items to privately answer a prediction query within a few seconds on a single PC, while its prediction accuracy still competitive with state-of-the-art recommender systems computing over clear data.

The research work we have performed during the course of the PhD program has lead to the following publications:

- Jun Wang, Afonso Arriaga, Qiang Tang, and Peter YA Ryan. "CryptoRec: Privacy-preserving Recommendations as a Service." arXiv preprint arXiv:1802.02432 (2018)

- Jun Wang, and Qiang Tang. "Differentially Private Neighborhood-based Recommender Systems." IFIP International Conference on ICT Systems Security and Privacy Protection. Springer, Cham, 2017.

- Jun Wang, and Qiang Tang. "A probabilistic view of neighborhood-based recommendation methods." Data Mining Workshops (ICDMW), 2016 IEEE 16th International Conference on. IEEE, 2016.

- Qiang Tang, and Jun Wang. ”Privacy-preserving friendship-based recommender systems.” IEEE Transactions on Dependable and Secure Computing (2016).

- Qiang Tang, and Jun Wang. ”Privacy-preserving context-aware recommender systems: Analysis and new solutions.” European Symposium on Research in Computer Security. Springer, Cham, 2015.

1.4 Organization

The structure of this dissertation is organized as follows. We lay down state-of-the-art recommendation methods (e.g., neighborhood-based methods and matrix factorization, etc.) and security primitives (e.g., homomorphic encryption and differential privacy), in Chapter 2. Next, we design and construct privacy-preserving solutions for recommender systems, under different scenarios (i.e., security models). In Chapter 3, we introduce a new solution to build differential privacy into neighborhood-based recommender systems. In Chapter 4, we exploit social context and a homomorphic encryption scheme to build an efficient privacy-preserving solution for accurately computing recommendations. In Chapter 5, we present a privacy-preserving framework for recommendation as a service, in which the key technical innovation is our proposed crypto-friendly recommender system. In chapter 6, we summarize this dissertation and introduce directions for the future work. For the sake of readability, each chapter in this dissertation is written to be as independent as possible, so that readers can dive into the chapters which interest them most without having to consult other chapters.

2
Background

In this chapter, we first introduce notations and frequently used abbreviations. Then we describe state-of-the-art recommendation algorithms. Lastly, we present the security primitives used to build privacy-preserving solutions.

2.1 Notations and Abbreviations

The notations and commonly used variables of this dissertation are summarized in Table 2.0.1. Scalars are denoted in lower-case characters, vectors are denoted in **lower-case bold** characters, matrices are denoted in **Upper-case bold** characters. We write $a \leftarrow b$ to denote the algorithmic action of assigning the value of b to the variable a. If there is no specific explanation, we will follow this rule. Frequently used terms and their corresponding abbreviations are summarized in Table 2.0.2.

n/m	number of users / items
$\mathbf{R} \in \mathbb{N}^{n\times m}$	rating matrix
r_{ui}	rating given by user u for item i
$\hat{r}_{ui}$	estimation of r_{ui}
$\mathbf{r}_u \in \mathbb{N}^{1\times m}$	rating vector of user u : $\{r_{ui}\}_{i=1}^{m}$
$\mathbf{r}_i \in \mathbb{N}^{n\times 1}$	rating vector of item i : $\{r_{ui}\}_{u=1}^{n}$
$\bar{r}_u/\bar{r}_i$	mean rating of user u / item i
φ_{ui}	$\varphi_{ui} = 1$ if r_{ui} exists, otherwise $\varphi_{ui} = 0$.
Θ	general form of model parameters
$[\![x]\!]$	encryption of x
$[\![\mathbf{x}]\!]$	$\{[\![x_1]\!], [\![x_2]\!], [\![x_3]\!], \cdots\}$
pk/sk	public key / secret key
$\oplus$	addition between two ciphertexts or a plaintext and a ciphertex
$\odot$	multiplication between a plaintext and a ciphertex
$\otimes$	multiplication between two ciphertexts

Table 2.0.1: Variables and notations

2.2 State-of-the-art Recommender Systems

Recommender systems have experienced a rapid development in the past decade. State-of-the-art recommender systems are often built upon collaborative filtering techniques which directly model user-item interactions. A number of reviews have comprehensively investigated this area [3, 25, 114, 132]. In order to introduce recommender systems clearly and briefly, as show in Figure 2.2.1, we classify existing recommendation algorithms into two branches, conventional approach and deep approach, according to the evolution of algorithms. In the conventional approach, neighborhood-based methods (NBMs) and matrix factorizaiton (MF) are two representative algorithms; With the success of deep learning in multiple areas such as computation version [116, 117] and speech recognition [51], applying deep learning to build recommender systems have been emerging, refer to as deep approach. Generally, there are two directions in the deep approach, one is using a single neural network such as autoencoder [106], convolutinal neural network to

CF	Collaborative Filtering
NBM	Neighborhood-based Method
MF	Matrix Factorization
SGD	Stochastic Gradient Descent
MCMC	Monto Carlo Markov Chain
SGLD	Stochastic Gradient Langevin Dynamics
GC	Garbled Circuits
HE	Homomorphic Encryption

Table 2.0.2: Abbreviations

build recommender systems; the other is a hybrid method which combines either different deep models or deep models and conventional recommendation models. In this section, we introduce typical recommendation models under the conventional approach and deep approach. It is worthy stressing that both the two approaches are important to recommender systems. We refer interested readers to investigations [25, 114, 132] for more details.

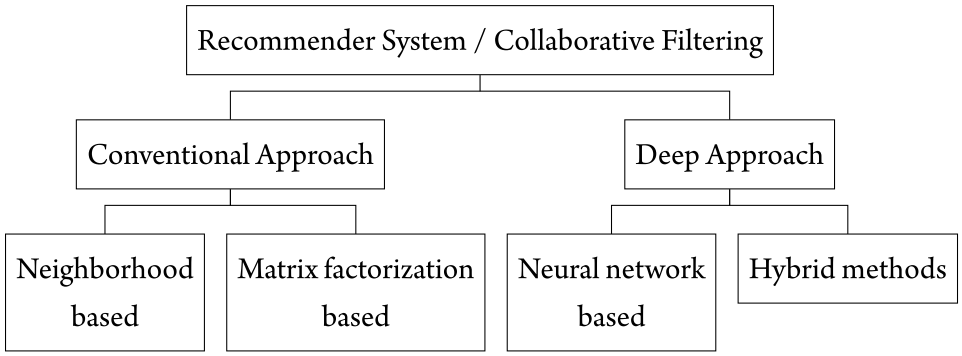


Figure 2.2.1: Recommender System

2.2.1 Neighborhood-based Methods

Neighborhood-based methods (NBMs) estimate a user's rating on a targeted item by taking the weighted average of a certain number of ratings of the user or of the item. The similarity between items or users often serve as the weights. Formally, we can describe an item-based NBM as follows,

$$\hat{r}_{ui} = \bar{r}_i + \frac{\sum_{j \in \mathcal{N}_u(i)} s_{ij}(r_{uj} - \bar{r}_j)}{\sum_{j \in \mathcal{N}_u(i)} |s_{ij}|}, \tag{2.1}$$

where $\bar{r}_i$ is the mean rating of item i, $s_{ij} \in \mathbf{S}^{m \times m}$ represents the similarity between item i and j, and $\mathcal{N}_u(i)$ denotes a set of items rated by user u that are the most similar to item i according to the similarity matrix $\mathbf{S} \in \mathbb{R}^{m \times m}$. It is clear that the similarity plays a critical role in the recommendation computation. Pearson correlation is one of the most widely used similarity metrics [114]:

$$s_{ij} = \frac{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_i)(r_{uj} - \bar{r}_j)}{\sqrt{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_i)^2} \sqrt{\sum_{u \in \mathcal{U}_{ij}} (r_{uj} - \bar{r}_j)^2}}, \tag{2.2}$$

where $\mathcal{U}_{ij}$ denotes the set of users that rated both items i and j.

Another approach is using regression method to learn similarity from data directly, which often leads to more accurate prediction results but pays the cost of computational complexity. In this approach, we first define an objective function as follows,

$$\mathcal{L} = \sum_{u=1}^{n} \sum_{i=1}^{m} (\hat{r}_{ui} - r_{ui})^2 + \lambda \cdot ||S||^2 \tag{2.3}$$

where $\hat{r}_{ui}$ is defined in Equation (2.1). The optimal similarity can be learned by minimizing the objective function.

User-based NBM (U-NBM) is the symmetric counterpart of I-NBM. Normally, I-NBM is more accurate and robust than U-NBM [114].

2.2.2 Matrix Factorization

Let $\mathbf{R}^{n \times m}$ be a sparse rating matrix formed by n users and m items, in which each user rated only a small number of the m items, and the missing values are marked with zero. Matrix factorization (MF) decomposes the rating matrix $\mathbf{R}$ into two low-rank and dense feature matrices [64]:

$$\mathbf{R} \approx \mathbf{P}\mathbf{Q}^T, \quad (2.4)$$

where $\mathbf{P} \in \mathbb{R}^{n\times d}$ is the user feature space, $\mathbf{Q} \in \mathbb{R}^{m\times d}$ is the item feature space and $d \in \mathbb{N}^+$ is the dimension of user and item features. To predict how user u would rate item i, we compute $\hat{r}_{ui} = \mathbf{p}_u\mathbf{q}_i^T$, where $\mathbf{p}_u^{1\times d} \subset \mathbf{P}$ and $\mathbf{q}_i^{1\times d} \subset \mathbf{Q}$ denote the learned features vectors of user u and item i, respectively. A standard way of optimizing $\mathbf{P}$ and $\mathbf{Q}$ is to minimize a regularized squared objective function

$$\overset{min}{\mathbf{P},\mathbf{Q}} \sum_{(u,i)\in\mathbf{R}} (\mathbf{p}_u\mathbf{q}_i^T - r_{ui})^2 + \lambda(||\mathbf{p}_u||^2 + ||\mathbf{q}_i||^2), \quad (2.5)$$

by using the stochastic gradient descent (SGD) optimization method [64], but only based on observed ratings (rating matrix $\mathbf{R}$ is sparse). The constant λ is a regularization factor.

2.2.3 Neural Network based Recommender Systems

In addition to the success of neural networks in visual recognition and speech synthesis tasks is widely diffused, many works also focus on constructing neural recommender systems. (We refer to the reader to [132] for an overview.) AutoRec [106] is a notable example, built on top of Autoencoders [85]. Item-based AutoRec (I-AutoRec) reconstructs the inputs $\mathbf{r}_i$ by computing

$$\hat{\mathbf{r}}_i = f(\mathbf{W} \cdot g(\mathbf{V}\mathbf{r}_i + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}), \quad (2.6)$$

where $g(\cdot)$ and $f(\cdot)$ are activation functions, e.g. the Sigmoid function ($\frac{1}{1+e^{-x}}$) or ReLu function ($max(0, x)$). Non-linear activation functions are crucial to the success of neural networks. Model parameters are defined as follows: $\Theta = \{\mathbf{W}, \mathbf{V}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}\}$, where $\mathbf{W} \in \mathbb{R}^{n\times d}$ and $\mathbf{V} \in \mathbb{R}^{d\times n}$ are for 'transformations', and $\mathbf{b}^{(1)} \in \mathbb{R}^{d\times 1}$ and $\mathbf{b}^{(2)} \in \mathbb{R}^{n\times 1}$ are for "bias" terms. Θ is learned by using the SGD to minimize the regularized square objective function

$$\overset{min}{\mathbf{W}, \mathbf{V}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}} \sum_{i \in R} ||\hat{\mathbf{r}}_i - \mathbf{r}_i||^2 + \lambda(||\mathbf{W}||^2 + ||\mathbf{V}||^2), \tag{2.7}$$

where the gradient of each model parameter is computed by only observed ratings [106]. Equation (2.7) defines I-AutoRec. The user-based AutoRec (U-AutoRec) is defined symmetrically in the obvious way. Experimental results show that I-AutoRec outperforms U-AutoRec in terms of accruracy [106].

Back-propagation [65] is a standard method used in neural networks to calculate a gradient that is needed in the calculation of the values of parameters in the network. Simply put, we first compute the predictions given the input data (i.e., a forward pass); and then we calculate the total error according to the objective function. Lastly, we compute the gradient of each trainable parameter using the error back-propagated through the network layers (i.e., a backward pass).

2.2.4 Training

In this section, we first introduce a general form of machine learning training process, then we present a specific training case that training matrix factorization using SGD.

2.2.4.1 A General Form

A machine learning training process often refers to that, given an objective function (e.g., Equation (2.3) and (2.5)) with regard to a specific machine learning model, using an iterative method to optimize the model parameters (a.k.a, weights) by minimizing the objective function, where the model parameters are often randomly initialized.

In machine learning tasks, the problem of minimizing an objective function often has the form of a sum:

$$\mathcal{E}(\Theta) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{E}_i(\Theta) \tag{2.8}$$

The parameter Θ is optimized by minimizing $\mathcal{E}(\Theta)$. Each summand function $\mathcal{E}_i(\Theta)$ is typically associated with the i-th example (or a batch of examples) in the training set. In this case, $\mathcal{E}_i(\Theta)$ is the value of the objective function at i-th example, and $\mathcal{E}(\Theta)$ is the empirical risk.

A standard approach of minimizing the objective function is to iteratively update the model parameter Θ in the opposite direction of their corresponding gradients,

$$\Theta \leftarrow \Theta - \eta\sum_{i=1}^{n}\frac{\Delta\mathcal{E}_i(\Theta)}{n} \tag{2.9}$$

where η is the learning rate which decides the speed of updates, $\Delta\mathcal{E}_i(\Theta) = \frac{\partial\mathcal{E}_i(\Theta)}{\partial\Theta}$ is gradient calculated from the i-th examples (or a batch of examples). As the training process sweeps through the whole training set, it performs the above update for each training example. Algorithm 1 outlines the training process. Based on this general form, stochastic gradient descent (SGD) and its variants, such as momentum SGD [61] and Adam [61], are the most commonly used machine learning training methods.

Algorithm 1 The outline of a typical training process

Input: training data $\mathcal{D}$, initial parameters Θ^1, learning rate η
Output: optimized Θ^T

```
1: procedure Train($\mathcal{D}, \Theta, \eta$)
2:     for $j \leftarrow \{1, 2, \cdots, T\}$ do                ▷ $T$: # of training iterations
3:         Randomly shuffle examples in the training set $\mathcal{D}$.
4:         for $i \leftarrow \{1, 2, \cdots, n\}$ do
5:             $\Theta^{j+1} \leftarrow \Theta^j - \eta \Delta \mathcal{E}_i(\Theta^j)$
6:     return $\Theta^T$
```

2.2.4.2 A Specific Training Case

In this section, we take matrix factorization as an intuitive example to show how to learn optimal user and item features via SGD.

Recalling the definition of matrix factorization in section 2.2.2, a sparse rating matrix **R** is decomposed into two dense low-rank metrics, the user features **P** and item features **Q**. The optimal features can be learned via SGD by minimizing the regularized squared objective function (also described in Equation (2.5)) as follows,

$$\overset{min}{\mathbf{P}, \mathbf{Q}} \sum_{(u,i)\in\mathbf{R}} (\mathbf{p}_u \mathbf{q}_i^T - r_{ui})^2 + \lambda(||\mathbf{p}_u||^2 + ||\mathbf{q}_i||^2), \tag{2.10}$$

We first compute gradient with regarding to each model parameters as follows,

$$\begin{aligned} \Delta \mathbf{q}_i &= e_{ui}\mathbf{p}_u - \lambda \mathbf{q}_i \\ \Delta \mathbf{p}_u &= e_{ui}\mathbf{q}_i - \lambda \mathbf{p}_u \end{aligned} \tag{2.11}$$

where λ is the regularization parameter.

The update rule of parameters is to move along the opposite direction of gradi-

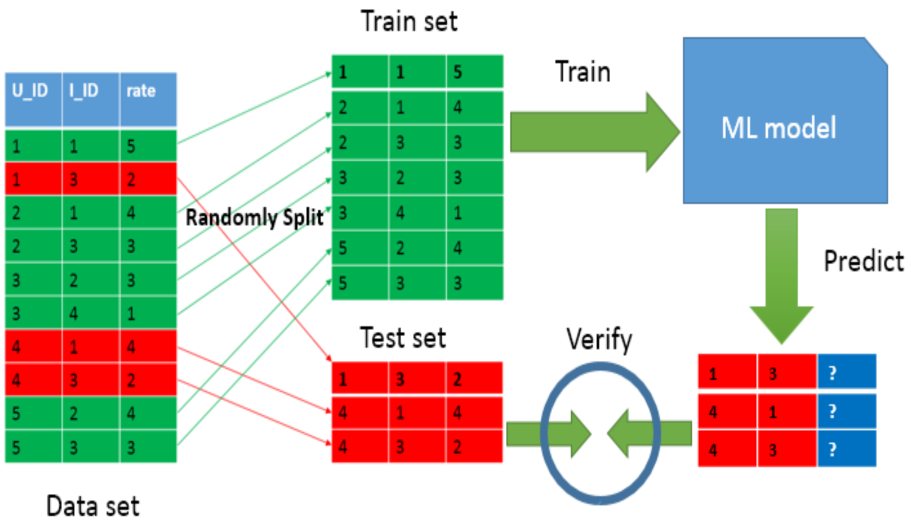


Figure 2.2.2: A typical cross validation process

ents. Therefore, we have

$$\begin{aligned} \mathbf{q}_i &\leftarrow \mathbf{q}_i - \eta \Delta \mathbf{q}_i \\ \mathbf{p}_u &\leftarrow \mathbf{p}_p - \eta \Delta \mathbf{p}_p \end{aligned} \tag{2.12}$$

The training process keeps updating the model parameters $(\mathbf{P}, \mathbf{Q})$ until the model converges (see Algorithm 1). The convergence is often evaluated by cross validation, which will be introduced in section 2.2.4.3.

2.2.4.3 Cross Validation

In a prediction task, a machine learning model is usually trained on a dataset of known data (i.e., training dataset), and tested by a dataset of unknown data (i.e., test dataset). Cross validation is one of the most widely adopted methods to assess how the results of a statistical analysis generated from the training set will generalize to the test dataset, as shown in Figure 2.2.2.

Among various cross validation methods, K-fold cross validation is a simple and widely used approach, summarized as follows,

1. Randomly dividing the original dataset into K equal-sized subsets. One of the K subset is selected as the test set and the other K-1 subsets serve as the training set.

2. Repeating the validation process K times, with each of the K subset used exactly once as the test data.

3. Computing and reporting the average error (regarding a specific metric) across all K trials.

The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly once.

2.2.5 Metrics

Generally, we can classify various recommendation tasks into two categories, rating prediction and top-n recommendation. The former estimates a user's exact preference on an item. The latter predicts top-n items that a user likes the most. Using off-line testing to measure the recommendation quality (i.e., accuracy) is crucial for successfully deploying recommender systems.

For **rating prediction** tasks, the root-mean-square-error (RMSE) and mean-absolute-error (MAE) are widely adopted as the accuracy performance metrics.

RMSE.

$$RMSE = \sqrt{\frac{\sum_{(u,i)\in\mathcal{D}}(\hat{r}_{ui} - r_{ui})^2}{|\mathcal{D}|}} \tag{2.13}$$

MAE.

$$MAE = \frac{\sum_{(u,i)\in\mathcal{D}}|\hat{r}_{ui} - r_{ui}|}{|\mathcal{D}|} \tag{2.14}$$

where $\mathcal{D}$ is the testing set, $|\mathcal{D}|$ is the number of ratings in the testing set. The lower the RMSE (MAE) value, the higher the accuracy performance is.

For **top-n recommendation**, *precision*, *recall* and normalized-discounted-cumulative-gain (*nDCG*) are widely employed to evaluate the accuracy performance.

precision@k.

$$precision = \frac{\#\ of\ recommended\ items\ @k\ that\ are\ relevant}{\#\ of\ recommended\ items\ @k} \tag{2.15}$$

The precision metric indicates the fraction of relevant items among the recommended items.

recall@k.

$$recall = \frac{\#\ of\ recommended\ items\ @k\ that\ are\ relevant}{\#\ of\ relevant\ items} \tag{2.16}$$

The recall metric is the fraction of relevant items that have been recommended over the total amount of relevant items.

Beside the accuracy related metrics, there are also some other important criteria such as diversity and novelty. For more detail, we refer readers to a more comprehensive investigation [47].

2.3 Homomorphic Encryption

Homomorphic encryption (HE) is a form of encryption that allows computations to be carried over ciphertexts. The result, after decryption, is the same as if the operations had been performed on the plaintexts [42]. As an illustrative example, consider two plaintexts x_1 and x_2 and their corresponding ciphertexts $[\![x_1]\!] \leftarrow \mathsf{Enc}(x_1, \mathsf{pk})$ and $[\![x_2]\!] \leftarrow \mathsf{Enc}(x_2, \mathsf{pk})$. An encryption scheme is additively homomorphic if it satisfies $x_1 + x_2 = \mathsf{Dec}([\![x_1]\!] \oplus [\![x_2]\!], \mathsf{sk})$ or multiplicatively homomorphic if we have $x_1 \times x_2 = \mathsf{Dec}([\![x_1]\!] \otimes [\![x_2]\!], \mathsf{sk})$, where $\oplus$ and $\otimes$ represent the homomorphic addition and homomorphic multiplication operations, respectively.

Some HE schemes are only either additively homomorphic or multiplicatively homomorphic, such as [91]. The schemes that fall into this category are know to be *partially homomorphic* (PHE). Schemes that support both additions and multiplications, but only a limited number of times, are known as *somewhat homomorphic* (SWHE), as opposed to those that allow an unbounded number of homomorphic operations, which are called *fully homomorphic encryption* (FHE) schemes [42]. The efficiency of the schemes in each class is usually related to the expressiveness of the supported operations, meaning that PHE schemes are more efficient than SWHE schemes, which in turn are more efficient that FHE schemes. For PHE, Paillier cryptosystem [91] is an often adopted scheme; For SWHE, leveled homomorphic encryptions are popular approaches [37, 48]. FHE often leads to a efficiency bottleneck, so that it is not often used in privacy-preserving machine learning tasks.

In addition to the additively or multiplicatively homomorphic properties of ciphertexts, HE schemes also allow additions and multiplications between a ciphertext and a plaintext, i.e. $x_1 + x_2 = \mathsf{Dec}([\![x_1]\!] \oplus x_2, \mathsf{sk})$ and $x_1 \times x_2 = \mathsf{Dec}([\![x_1]\!] \odot x_2, \mathsf{sk})$.

Syntax.. A homomorphic encryption scheme is a tuple of four ppt algorithms $\mathsf{HE} := (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ as follows:

- $\mathsf{Setup}(1^\lambda)$ is the setup algorithm. It takes as input the security parameter λ and outputs a private/public key pair $(\mathsf{sk}, \mathsf{pk})$. The public key pk includes a description of the message space $\mathcal{M}$.

- $\mathsf{Enc}(\mathsf{m}, \mathsf{pk})$ is the encryption algorithm, which takes as input the public key pk and a message $\mathsf{m} \in \mathcal{M}$ and outputs a ciphertext c.

- $\mathsf{Eval}(f, \mathsf{c}_1, \ldots, \mathsf{c}_t, \mathsf{pk})$ is the homomorphic evaluation algorithm. It takes as input a public key pk, a circuit $f : \mathcal{M}^t \to \mathcal{M}$ in a class $\mathcal{F}$ of supported circuits and t ciphertexts $\mathsf{c}_1, \ldots, \mathsf{c}_t$, and returns a ciphertext c.

- $\mathsf{Dec}(\mathsf{c}, \mathsf{sk})$ is the decryption algorithm that on input a secret key sk and a ciphertext c, it returns a message m or a special failure symbol $\perp$.

2.4 Differential Privacy

It is impossible to release information from a private statistical database without revealing any private information, —there is no free lunch for privacy and utility [26]. Differential privacy[33] aims to provide rigorous means to maximize the accuracy of answering the queries of statistical databases, while quantifying and minimizing the privacy loss resulted to individuals when their private information are used in the creation of a data product.

Definition 1 (Differential Privacy). *A randomized algorithm $\mathcal{F}$ with domain $\mathbb{N}^{|\mathcal{X}|}$ is (ε, σ)-differentially private if for all $\mathcal{O} \subset Range(\mathcal{F})$ and for any possible database pair $(\mathcal{D}_0, \mathcal{D}_1)$ satisfies:*

$$Pr(\mathcal{F}(\mathcal{D}_0) \in \mathcal{O}) \leq e^{\varepsilon} Pr(\mathcal{F}(\mathcal{D}_1) \in \mathcal{O}) + \sigma$$

where databases $(\mathcal{D}_0, \mathcal{D}_1)$ differ on only one record.

If $\varepsilon = 0$, we say that $\mathcal{F}$ is ε-differentially private. With this definition, the larger the value of ε, the larger the privacy loss is. Simply put, differential privacy addresses the case that when a trusted data curator wants to release some statistical information over its database without revealing the particular value of a record. It allows a data owner to deny that he (or she) has participated in a computation.

2.4.1 Important Properties

In this section, we introduce two important properties, Post-processing and Sequential Composition Theory, which are fundamental tools to provide theoretical supports for constructing differentially private machine learning models.

2.4.1.1 Post-processing

Theorem 1 (Post-processing). *Let $\mathcal{F} : \mathbb{N}^{|\mathcal{X}|} \rightarrow \mathbb{R}$ be an random algorithm that is (ε, σ)-differentially private. Let $\mathcal{F}' : \mathbb{R} \rightarrow \mathbb{R}'$ be an arbitrary randomized mapping. Then $\mathcal{F}' \circ \mathcal{F} : \mathbb{N}^{|\mathcal{X}|} \rightarrow \mathbb{R}'$ is (ε, σ)-differentially private.*

This property indicates that a malicious user, without learning additional knowledge about a private database, cannot compute a function of the output of a private algorithm $\mathcal{F}$, compromising its differential privacy guarantee.

2.4.1.2 Sequential Composition Theory

Theorem 2 (Sequential Composition Theory). *Let $\mathcal{F}_i : \mathbb{N}^{|\mathcal{X}|} \rightarrow \mathbb{R}_i$ be an $(\varepsilon_i, \sigma_i)$-differentially private algorithm, then $\mathcal{F}_{[k]} \rightarrow \prod_{i=1}^{k} \mathbb{R}_i$ is $(\sum_{i=1}^{k} \varepsilon_i, \sum_{i=1}^{k} \sigma_i)$-differentially private, where $\mathcal{F}_{[k]}(\mathcal{D})$ denotes $[\mathcal{F}_1(\mathcal{D}), \mathcal{F}_2(\mathcal{D}), \cdots, \mathcal{F}_k(\mathcal{D})]$*

With regarding to training a machine learning model, Training methods, such as SGD, will access training data multiple times before the model converges. The composition theory, in conjunction with the post-process property, guarantees that the differential privacy loss linearly increases with the number of accesses to the training database.

2.4.2 Implementations

In this section, we introduce Laplace Mechanism [32] and Exponential mechanism[77], which are two typical approaches for implementing differentially private solutions.

2.4.2.1 Laplace Mechanism

Any differentially private mechanism is necessarily randomized, as differential privacy is a probabilistic concept. Adding noise is a common approach to build randomization into algorithms. The Laplace mechanism, as its name suggests, it adds noise sampled from Laplace distribution. The Laplace mechanism is defined as following,

$$\mathcal{F}(\mathcal{D}) = f(\mathcal{D}) + \mathcal{Y}, \ \mathcal{Y} \sim Lap(-\frac{|x|}{\lambda}) \tag{2.17}$$

where f is the original real valued query/algorithm we planned to execute on a statistical database $\mathcal{D}$. With this definition, the output of $\mathcal{F}(\mathcal{D})$ can be considered to be a random variable, where for $t \in \mathbb{R}$, we have,

$$\frac{Pr(\mathcal{F}(\mathcal{D}_0) = t)}{Pr(\mathcal{F}(\mathcal{D}_1) = t)} = \frac{Lap(t - f(\mathcal{D}_0))}{Lap(t - f(\mathcal{D}_1))} \tag{2.18}$$

Since the noise is sampled from Laplace distribution, therefore,

$$\begin{aligned}\frac{Pr(\mathcal{F}(\mathcal{D}_0) = t)}{Pr(\mathcal{F}(\mathcal{D}_1) = t)} &= \frac{Lap(t - f(\mathcal{D}_0))}{Lap(t - f(\mathcal{D}_1))} \\ &= e^{\frac{|f(\mathcal{D}_0) - f(\mathcal{D}_1)|}{\lambda}} \\ &\leq e^{\frac{\Delta f}{\lambda}}\end{aligned} \tag{2.19}$$

According to the definition of differential privacy, if $e^{\frac{\Delta f}{\lambda}} = e^{\varepsilon}$ (i.e., $\mathcal{Y} \leftarrow_{\$} Lap(-\frac{\Delta f}{\lambda})$), then the Laplace mechanism guarantees ε-differential privacy. Δf measures the largest change caused by a single difference between the database $\mathcal{D}_0$ and $\mathcal{D}_1$. In the Laplace mechanism, Δf is captured by l_1 distance, referred to as l_1-*sensitivity*.

Definition 2 (l_1-sensitivity). *The l_1-sensitivity of a function f: $\mathbb{N}^{|\mathcal{D}|} \rightarrow \mathbb{R}^k$ is,*

$$\Delta f = max||f(\mathcal{D}_0) - f(\mathcal{D}_1)||_1$$

2.4.2.2 Exponential Mechanism

The Laplace mechanism provides differential privacy to real-valued functions. The exponential mechanism helps to extend the notion of differential privacy to a more generic mechanism, that the problem of non-numeric queries can be also addressed.

Consider a general setting which maps a set of finite elements of domain $\mathcal{D}$ to a range $\mathcal{R}$, if each element of the domain $\mathcal{D}$ corresponds to the probability distribution over the range $\mathcal{R}$, the mapping can be randomized. The exponential mechanism is built on this ground truth, and it defines a utility function, $q = \mathcal{D} \times \mathcal{R} \rightarrow \mathbb{R}$, to measure the quality of the pair (d, r), where $d \in \mathcal{D}$ and $r \in \mathcal{R}$. The function $q(d, r)$ is monotonically increasing with regarding the quality of (d, r).

The exponential mechanism $\mathcal{F}$ outputs $r \in \mathcal{R}$ with probability proportional to

$$Pr(r) \sim exp(\frac{\varepsilon q(d, r)}{2\Delta q}) \tag{2.20}$$

can guarantee $(\varepsilon, 0)$-differential privacy, where Δq defines the maximum change of the function $q(d, r)$ caused by a single difference of the inputs, as known as *sen-*

sitivity. It is defined as

$$\Delta q = r \overset{max}{\in} \mathcal{R} \, ||\mathcal{D}_0 - \overset{max}{\mathcal{D}_1}||_1 \leq 1 \quad ||q(\mathcal{D}_0, r) - q(\mathcal{D}_1, r)||_1 \tag{2.21}$$

The density of $\mathcal{F}(d)$ at r is equal to

$$\frac{exp(\frac{\varepsilon q(\lceil, r)}{2\Delta q})}{\sum exp(\frac{\varepsilon q(\lceil, r')}{2\Delta q})} \tag{2.22}$$

Clearly, a single change in $\mathcal{D}$ can change q by at most Δq, giving a factor of at most $exp(\frac{\varepsilon}{2})$ in the numerator and at least $exp(\frac{\varepsilon}{2})$ in the denominator, giving $exp(\varepsilon)$. Therefore, the exponential mechanism $\mathcal{F}$ is $(\varepsilon, 0)$-differentially private.

3

Differentially Private Recommender Systems

3.1 Introduction

Recommender systems often adopt a client-server model, in which a single server (or a cluster of servers) holds a database and serves a large number of users. State-of-the-art recommendation algorithms, such as matrix factorizaiton [63] and neighborhood based methods (NBMs) [114], exploit the fact that similar users are likely to prefer similar products, unfortunately this property also facilitates effective user de-anonymization and history information recovery through the recommendation results [16, 82]. Compared to matrix factorization, NBM is more fragile (e.g. [16, 79]), since it is essentially a simple linear combination of user history data which is weighted by the normalized similarity between users or items. To prevent user privacy from inference attack (i.e., analyzing prediction results), differential privacy [33] is a dominate approach. Intuitively, it offers a participant a possibility to deny her participation in a computation. Mcsherry et al. [76] built differential privacy into neighborhood-based recommender system, where they differential-privately calculate correlation as the similarity between users or items. However, using correlation as similarity often leads to a solution not as accurate as matrix factorization. Regression-based NBMs (i.e., learning similarity by regression methods) are able to improve accuracy. However, the regression-based approach often brings difficulty in deploying differential privacy, which has been demonstrated in building other differentially private regression-based models such as matrix factorization [9, 73].

Our Contributions. We aim to build differential privacy into regression-based neighborhood recommendation algorithms, thus to achieve a better trade-off between accuracy and privacy. Our contributions come from two aspects as follows,

- We present a general probabilistic graphical model for the family of neighborhood based methods, referred to as probabilistic neighborhood-based method (PNBM). PNBM allows us to explore building differentially private neighborhood-based recommender systems in different directions.
  - PNBM can lead user preference estimation to a problem of searching the maximum a posteriori similarity. By this, we can calibrate noise into the training process (i.e. SGD) to guarantee differential privacy.
  - PNBM can link the differential privacy concept to NBMs, by sampling similarity from scaled posterior distribution. For the sake of efficiency, we employ a recent MCMC method, Stochastic Gradient Langevin Dynamics (SGLD) [126], as the sampler. In order to use SGLD, we derive an unbiased estimator of similarity gradient from a mini-batch.
- We carry out experiments, on two real world datasets, to compare our solutions to the state-of-the-art differentially private solutions for matrix fac-

torization, and also to compare our solutions between themselves. Our results show that differentially private matrix factorization are more accurate when privacy loss is large (extremely, in a non-private case), but differentially private NBMs are better when privacy loss is set in a more reasonable range. Even with the added noises, both our solutions consistently outperform non-private traditional NBMs in accuracy.

3.2 A Probabilistic View of Neighborhood-based Methods

In this section, we extend the family of neighborhood-based methods into a general probabilistic graphical model. Before going to the details, we first review a typical definition of neighborhood-based method as follows,

$$\hat{r}_{ui} = \bar{r}_i + \frac{\sum_{j \in \mathcal{N}_u(i)} s_{ij}(r_{uj} - \bar{r}_j)}{\sum_{j \in \mathcal{N}_u(i)} |s_{ij}|}, \tag{3.1}$$

where $\bar{r}_i$ is the mean rating of item i, $s_{ij} \in \mathbf{S}^{m \times m}$ represents the similarity between item i and j, and $\mathcal{N}_u(i)$ denotes a set of items rated by user u that are the most similar to item i according to the similarity matrix $\mathbf{S} \in \mathbb{R}^{m \times m}$. It is clear that the similarity plays a critical role in the recommendation computation. For simplicity, we abstractly describe the above neighborhood-based method definition as a general function which takes as inputs similarity $\mathbf{S}$ and ratings $\mathbf{R}$:

$$\hat{r}_{ui} = f(\mathbf{s}_i, \mathbf{r}_u) \tag{3.2}$$

where $\mathbf{s}_i \subset \mathbf{S}$ denotes the similarity vector of item i, and $\mathbf{r}_u \subset \mathbf{R}$ is the rating vector of user u.

We assume that observed ratings $\mathbf{R}^{>0}$ conditioned on historical ratings with Gaussian noise $\mathcal{G}$, which leads to a probabilistic graphical model as shown in Figure 3.2.1. Formally, we define the likelihood function of all the observations $R^{>0}$

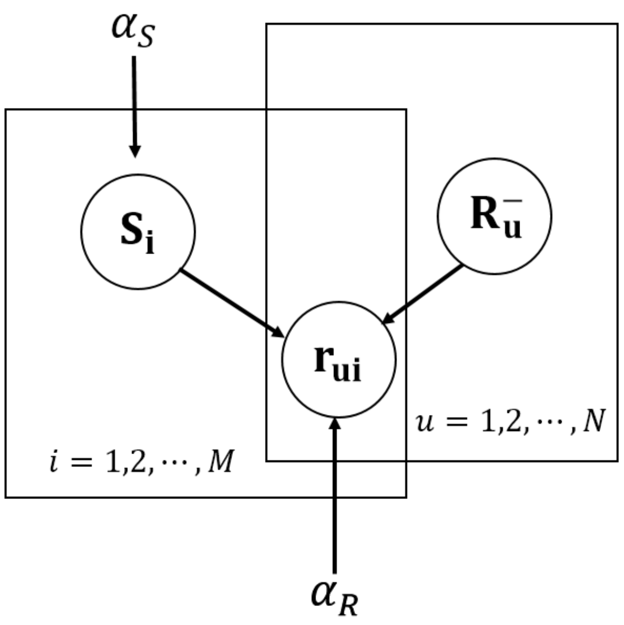


Figure 3.2.1: The graphical model of PNBM

and the prior of *similarity* **S** as

$$p(\mathbf{R}^{>0}|\mathbf{S}, \mathbf{R}^{-}, \alpha_R) = \prod_{i=1}^{M}\prod_{u=1}^{N}[\mathcal{G}(r_{ui}|f(\mathbf{s}_i, \mathbf{r}_u^{-}), \alpha_{\mathbf{R}}^{-1})]^{I_{ui}} \tag{3.3}$$

$$p(S|\alpha_{\mathbf{S}}) = \prod_{i=1}^{M}\mathcal{G}(\mathbf{s}_i|0, \alpha_{\mathbf{S}}^{-1}\mathbf{I}) \tag{3.4}$$

where $\mathcal{N}(x|\mu, \alpha^{-1})$ denotes the Gaussian distribution with mean μ and precision α. R^- indicates that if item i is being modeled then it is excluded from the training data $R^{>0}$. $f(S_i, R_u^-)$ denotes any NBM which takes as inputs the S_i and R_u^-. I_{uj} is the rating indicator $I_{uj} = 1$ if user u rated item j, otherwise, $I_{uj} = 0$.

With Equation (3.3) and (3.4), we can get the log of posterior distribution over the similarity as follows,

$$\begin{aligned}
-\log p(S|R^{>0}, \alpha_S, \alpha_R) &= -\log p(R^{>0}|S, R^-, \alpha_R)p(S|\alpha_S) \\
&= \frac{\alpha_R}{2}\sum_{i=1}^{m}\sum_{u=1}^{n}(r_{ui} - f(\mathbf{s}_i, \mathbf{r}_u))^2 \\
&+ \frac{\alpha_s}{2}\sum_{i=1}^{m}(||S_i||_2) + m^2\log\frac{\alpha_s}{\sqrt{2\pi}} + \log\frac{\alpha_R}{\sqrt{2\pi}}\sum_{i=1}^{m}\sum_{u=1}^{n} I_{ui}
\end{aligned} \tag{3.5}$$

3.2.1 Training

With the posterior distribution over the similarity (i.e., Equation (3.5)), estimating user preferences becomes of a problem of risk minimization. Generally, we have two approaches, i.e., learning similarity via Stochastic Gradient Descent and sampling similarity via Monte Carlo Markov Chain, to solve this problem. Therefore, we can explore building the differentially private recommender system from two directions.

- *Stochastic Gradient Descent (SGD).* In this approach, $\log p(S|R^{>0}, \alpha_S, \alpha_R)$ is treated as an error function. SGD can be adopted to minimize the error function. In each SGD iteration we update the gradient of *similarity* $(-\frac{\partial \log p(S|R^{>0}, \alpha_S, \alpha_R)}{\partial S_{ij}})$ with a set of randomly chosen ratings Φ by

$$S_{ij} \leftarrow S_{ij} - \eta(\sum_{(u,j)\in\Phi}(\hat{r}_{ui} - r_{ui})\frac{\partial \hat{r}_{ui}}{\partial S_{ij}} + \lambda S_{ij}) \tag{3.6}$$

  where η is the learning rate, $\lambda = \frac{\alpha_S}{\alpha_R}$ is the regular parameter, the set Φ may contain $n \in [1, N]$ users. In Section 3.3, we will introduce how to build the differentially private SGD to train probabilistic NBM.

- *Monte Carlo Markov Chain (MCMC).* We estimate the predictive distribution of an unknown rating by a Monte Carlo approximation. In Section 3.4, we will connect differential privacy to samples from the posterior $p(S|R^{>0}, \alpha_S, \alpha_R)$, via Stochastic Gradient Langevin Dynamics (SGLD) [126].

3.3 Differentially Private SGD

In this section, by leveraging the tight characterization of training data, NBM and SGD, we directly calibrate noise into the SGD training process, via Laplace mechanism, to differential-privately learn *similarity*. Algorithm 2 outlines our differentially-private SGD method for training probabilistic NBM.

Algorithm 2 Differentially Private SGD

```
Require: Database $R^{>0}$, privacy parameter $\varepsilon$, regular parameter $\lambda$, rescale parameter $\beta$, learning rate $\eta$, the total number of iterations $K$, initialized similarity $S^{(1)}$.
1: $S^{(1)} = S^{(1)} \cdot \beta$                                   ▷ rescale the initialization
2: for $t = 1 : K$ do
3:     • uniform-randomly sample a mini-batch $\Phi \subset R^{>0}$.
4:     $\Delta\mathcal{F} = 2e_{max}\frac{\tau}{C}$                     ▷ $e_{max} = 0.5 + \frac{\phi-1}{t+1}$; $|S_i|I_u^- \geq C$
5:     $e_{ui} = min(max(e_{ui}, -e_{max}), e_{max})$                  ▷ $e_{ui} = \hat{r}_{ui} - r_{ui}$
6:     $\mathbb{G} = \sum_{(u,i)\in\Phi} e_{ui}\frac{\partial\hat{r}_{ui}}{\partial S_i} + Laplace(\frac{\gamma K \Delta\mathcal{F}}{\varepsilon})$   ▷ $\gamma = \frac{L}{\mathcal{L}}$
7:     $S^{(t+1)} \leftarrow S^{(t)} - \eta(\beta\mathbb{G} + \lambda S^{(t)})$   ▷ up-scale the update
8: return $S^{(t+1)}$
```

According to Equation (3.5) and (3.6), for each user u (in a randomly chosen mini-batch Φ) the gradient of *similarity* is

$$\mathcal{G}_{ij}(u) = e_{ui}\frac{\partial\hat{r}_{ui}}{\partial S_{ij}} = e_{ui}\left(\frac{r_{uj}}{S_i I_u^-} - \hat{r}_{ui}\frac{I_{uj}}{S_i I_u^-}\right) \tag{3.7}$$

where $e_{ui} = \hat{r}_{ui} - r_{ui}$. For the convenience of notation, we omit $S_{ij} < 0$ part in Equation (3.7) which does not compromise the correctness of bound estimation.

To achieve differential privacy, we update the gradient $\mathcal{G}$ by adding Laplace noise (Algorithm 2, line 6). The amount of noise is determined by the bound of gradient $\mathcal{G}_{ij}(u)$ (sensitivity $\Delta\mathcal{F}$) which further depends on e_{ui}, $(r_{uj} - \hat{r}_{ui}I_{uj})$ and

$|S_i|I_u^-$. We reduce the sensitivity by exploiting the characteristics of training data, NBM and SGD respectively, by the following tricks.

Preprocessing is often adopted in machine learning for utility reasons. In our case, it can contribute to privacy protection. For example, we only put users who have more than 20 ratings in the training data. It results in a bigger $|S_i|I_u^-$ thus will reduce sensitivity. Suppose the rating scale is $[r_{min}, r_{max}]$, removing "paranoid" records makes $|r_{uj} - \hat{r}_{ui}I_{uj}| \leq \phi$ hold, where $\phi = r_{max} - r_{min}$.

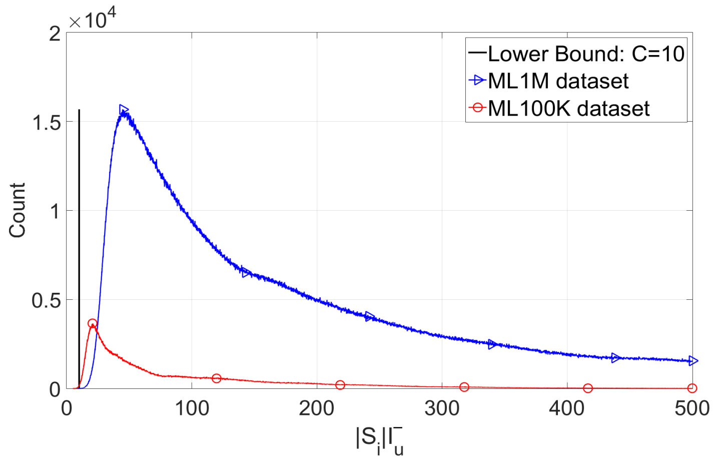


Figure 3.3.1: The distribution of $|S_i|I_u^-$ ($\beta = 10$). In order to have more detail of the distribution of those points have low $|S_i|I_u^-$ values, the points $|S_i|I_u^- \geq 500$ are removed.

Rescaling the value of similarity allows a lower sensitivity. NBM, Equation (3.1), allows us to rescale the *similarity* **S** to an arbitrarily large magnitude such that we can further reduce the sensitivity (by increasing the value of $|S_i|I_u^-$). However, the initialization of *similarity* strongly influences the convergence of the training. Thus, it is important to balance the convergence (accuracy) and the value of *similarity* (privacy). Another observation is that the gradient down-scales when enlarging the *similarity*, see Equation (3.7). We can up-scale the gradient monotonically during the training process (Algorithm 2, line 1 and 7). Fig. 3.3.1 shows , let $\beta = 10$, the lower bound of $|S_i|I_u^-$, denote as C, is 10.

The prediction error $e_{ui} = \hat{r}_{ui} - r_{ui}$ decreases when the training goes to convergence such that we can clamp e_{ui} to a lower bound dynamically. In our experiments, we bound the prediction error as $|e_{ui}| \leq 0.5 + \frac{\phi - 1}{t+1}$, where t is the iteration index. This constraint trivially influences the convergence under non-private training process.

After applying all the tricks, we have the dynamic gradient bound at iteration t as follows

$$max(|\mathcal{G}^{(t)}|) \leq (0.5 + \frac{\phi - 1}{t+1})\frac{\phi}{C} \tag{3.8}$$

The *sensitivity of* each iteration is $\Delta\mathcal{F} = 2max(|\mathcal{G}^{(t)}|) \leq 2(0.5 + \frac{\phi-1}{t+1})\frac{\phi}{C}$.

Theorem 3. *Uniform-randomly sample L examples from a dataset of the size $\mathcal{L}$, Algorithm 2 achieves ε-differential privacy if in each SGD iteration t we set $\varepsilon^{(t)} = \frac{\varepsilon}{K\gamma}$ where K is the number of iterations and $\gamma = \frac{L}{\mathcal{L}}$.*

Proof. In Algorithm 2, suppose the number of iterations K is known in advance, and each SGD iteration maintains $\frac{\varepsilon}{K\gamma}$-differential privacy. The privacy enhancing technique [8, 58] indicates that given a method which is ε-differentially private over a deterministic training set, then it maintains $\gamma\varepsilon$-differential privacy with respect to a full database if we uniform-randomly sample training set from the database where γ is the sampling ratio. Finally, combining the privacy enhancing technique with composition theory [33], it ensures the K iterations SGD process maintain the overall bound of ε-differential privacy. □

3.4 Differentially Private Posterior Sampling

Sampling from the posterior distribution of a Bayesian model with bounded log-likelihood has free differential privacy to some extent [124]. Specifically, for probabilistic NBM, releasing a sample of the *similarity* **S**,

$$S \sim p(S|R^{>0}, \alpha_S, \alpha_R) \propto exp(\sum_{i=1}^{M}\sum_{u=1}^{N}(r_{ui} - \frac{S_i R_u^-}{|S_i| I_u^-})^2 + \lambda \sum_{i=1}^{M} ||S_i||_2) \tag{3.9}$$

achieves $4B$-differential privacy at user level, if each user's log-likelihood is bounded to B, i.e. $\underset{u \in R^{>0}}{max} \sum_{i \in R_u} (\hat{r}_{ui} - r_{ui})^2 \leq B$. Wang et al. [124] showed that we can achieve ε-differential privacy by simply rescaling the log-posterior distribution with $\frac{\varepsilon}{4B}$, i.e. $\frac{\varepsilon}{4B} \cdot \log p(S|R^{>0}, \alpha_S, \alpha_R)$.

Posterior sampling is computationally costly. For the sake of efficiency, we adopt a recent introduced Monte Carlo method, Stochastic Gradient Langevin Dynamics (SGLD) [126], as our MCMC sampler. To successfully use SGLD, we need to derive an unbiased estimator of *similarity* gradient from a mini-batch which is a non-trivial task.

Next, we first overview the basic principles of SGLD (Section 3.4.1), then we derive an unbiased estimator of the true *similarity* gradient (Section 3.4.2), and finally present our privacy-preserving algorithm (Section 3.4.3).

3.4.1 Stochastic Gradient Langevin Dynamics

SGLD is an annealing of SGD and Langevin dynamics [102] which generates samples from a posterior distribution. Intuitively, it adds an amount of Gaussian noise calibrated by the step sizes (learning rate) used in the SGD process, and the step sizes are allowed to go to zero. When it is far away from the basin of convergence, the update is much larger than noise and it acts as a normal SGD process. The update decreases when the sampling approaches to the convergence basin such that the noise dominated, and it behaves like a Brownian motion. SGLD updates the candidate states according to the following rule.

$$\Delta\theta_t = \frac{\eta_t}{2}(\Delta \log p(\theta_t) + \frac{\mathcal{L}}{L}\sum_{i=1}^{L} \Delta \log p(x_{ti}|\theta_t)) + z_t; \quad z_t \sim \mathcal{N}(0, \eta_t) \quad (3.10)$$

where η_t is a sequence of step sizes. $p(x|\theta)$ denotes conditional probability distribution, and θ is a parameter vector with a prior distribution $p(\theta)$. L is the size of a mini-batch randomly sampled from dataset $\mathcal{X}^{\mathcal{L}}$. To ensure convergence to a local optimum, the following requirements of step size η_t have to be satisfied:

$$\sum_{t=1}^{\infty} \eta_t = \infty \qquad \sum_{t=1}^{\infty} \eta_t^2 < \infty$$

Decreasing step size η_t reduces the discretization error such that the rejection rate approaches zero, thus we do not need accept-reject test. Following the previous works, e.g. [73, 126], we set step size $\eta_t = \eta_1 t^{-\xi}$, commonly, $\xi \in [0.3, 1]$. In order to speed up the burn-in phase of SGLD, we multiply the step size η_t by a temperature parameter ϱ ($0 < \varrho < 1$) where $\sqrt{\varrho \cdot \eta_t} \gg \eta_t$ [23].

3.4.2 Unbiased Estimator of The Gradient

The log-posterior distribution of *similarity* **S** has been defined in Equation (3.5). The true gradient of the *similarity* **S** over $R^{>0}$ can be computed as

$$\mathcal{G}(R^{>0}) = \sum_{(u,i)\in R^{>0}} g_{ui}(S; R^{>0}) + \lambda S \tag{3.11}$$

where $g_{ui}(S; R^{>0}) = e_{ui} \frac{\partial \hat{r}_{ui}}{\partial S_i}$. To use SGLD and make it converge to true posterior distribution, we need an unbiased estimator of the true gradient which can be computed from a mini-batch $\Phi \subset R^{>0}$. Assume that the size of Φ and $R^{>0}$ are L and $\mathcal{L}$ respectively. The stochastic approximation of the gradient is

$$\mathcal{G}(\Phi) = \mathcal{L}\bar{g}(S, \Phi) + \lambda S \circ \mathbb{I}[i, j \in \Phi] \tag{3.12}$$

where $\bar{g}(S, \Phi) = \frac{1}{L} \sum_{(u,i)\in\Phi} g_{ui}(S, \Phi)$. $\mathbb{I} \subset \mathbb{B}^{M\times M}$ is symmetric binary matrix, and $\mathbb{I}[i, j \in \Phi] = 1$ if any item-pair (i, j) exists in Φ, otherwise 0. $\circ$ presents element-wise product (i.e. Hadamard product). The expectation of $\mathcal{G}(\Phi)$ over all possible mini-batches is,

$$\begin{aligned}\mathbb{E}_\Phi[\mathcal{G}(\Phi)] &= \mathbb{E}_\Phi[\mathcal{L}\bar{g}(S, \Phi)] + \lambda\mathbb{E}_\Phi[S \circ \mathbb{I}[i, j \in \Phi]] \\ &= \sum_{(u,i)\in R^{>0}} g_{ui}(S; R^{>0}) + \lambda\mathbb{E}_\Phi[S \circ \mathbb{I}[i, j \in \Phi]]\end{aligned} \tag{3.13}$$

$\mathbb{E}_\Phi[\mathcal{G}(\Phi)]$ is not an unbiased estimator of the true gradient $\mathcal{G}(R^{>0})$ due to the prior term $\mathbb{E}_\Phi[S \circ \mathbb{I}[i,j \in \Phi]]$. Let $\mathbb{H} = \mathbb{E}_\Phi[\mathbb{I}[i,j \in \Phi]]$, we can remove this bias by multiplying the prior term with $\mathbb{H}^{-1}$ thus to obtain an unbiased estimator. Follow previous approach [4], we assume the mini-batches are sampled with replacement, then $\mathbb{H}$ is,

$$\mathbb{H}_{ij} = 1 - \frac{|I_i||I_j|}{\mathcal{L}^2}(1 - \frac{|I_j|}{\mathcal{L}})^{L-1}(1 - \frac{|I_i|}{\mathcal{L}})^{L-1} \tag{3.14}$$

where $|I_i|$ (resp. $|I_j|$) denotes the number of ratings of item i (resp. j) in the complete dataset $R^{>0}$. Then the SGLD update rule is the following:

$$S^{(t+1)} \leftarrow S^{(t)} - \frac{\eta_t}{2}(\mathcal{L}\bar{g}(S^{(t)}, \Phi) + \lambda S^{(t)} \circ \mathbb{H}^{-1}) + z_t \tag{3.15}$$

3.4.3 Differential Privacy via Posterior Sampling

To construct a differentially private NBM, we exploit a recent observation that sampling from scaled posterior distribution of a Bayesian model with bounded log-likelihood can achieve ε-differential privacy [124]. We summarize the differentially private sampling process (via SGLD) in Algorithm 3.

Algorithm 3 Differentially Private Posterior Sampling (via SGLD)

```
Require: Temperature parameter $\varrho$, privacy parameter $\varepsilon$, regular parameter $\lambda$, initial learning rate $\eta_1$. Let $K$ larger than burn-in phase.
1: for $t = 1 : K$ do
2:     • Randomly sample a mini-batch $\Phi \subset R^{>0}$.
3:     $\bar{g}(S^{(t)}, \Phi) = \frac{1}{L}\sum_{(u,i)\in\Phi} e_{ui}\frac{\partial \hat{r}_{ui}}{\partial S_i^{(t)}}$        ▷ gradient of $\mathbf{S}$ (mini-batch)
4:     $z_t \sim \mathcal{N}(0, \varrho \cdot \eta_t)$        ▷ $\sqrt{\varrho \cdot \eta_t} \gg \eta_t$
5:     $S^{(t+1)} \leftarrow S^{(t)} - \frac{\varepsilon}{4B} \cdot \frac{\eta_t}{2}(\mathcal{L}\bar{g}(S^{(t)}, \Phi) + \lambda S^{(t)} \circ \mathbb{H}^{-1}) + z_t$
6:     $\eta_{t+1} = \frac{\eta_1}{t^\gamma}$
7: return $S^{(t+1)}$
```

Now, a natural question is how to determine the log-likelihood bound B? ($\underset{u \in R^{>0}}{max} \sum_{i \in R_u}(\hat{r}_{ui} - r_{ui})^2 \leq B$, and see Equation (3.9)). Obviously, B depends on the max rating number per user. To those users who rated more than τ items, we randomly remove some ratings thus to ensure that each user at most has τ ratings. In our context, the rating scale is [1,5], let $\tau = 200$, we have $B = (5-1)^2 \times 200$ (In reality, most users have less than 200 ratings [73]).

Theorem 4. *Algorithm 3 provides $(\varepsilon, (1+e^{\varepsilon})\delta)$-differential privacy guarantee to any user if the distribution $P'_{\mathcal{X}}$ where the approximate samples from is δ-far away from the true posterior distribution $P_{\mathcal{X}}$, formally $||P'_{\mathcal{X}} - P_{\mathcal{X}}||_1 \leq \delta$. And $\delta \to 0$ if the MCMC sampling asymptotically converges.*

Proof. Essentially, differential privacy via posterior sampling [124] is an exponential mechanism [77] which protects ε-differential privacy when releasing a sample θ with probability proportional to $exp(-\frac{\varepsilon}{2\Delta\mathcal{F}}p(\mathcal{X}|\theta))$, where $p(\mathcal{X}|\theta)$ serves as the utility function. If $p(\mathcal{X}|\theta)$ is bounded to B, we have the sensitivity $\Delta\mathcal{F} \leq 2B$. Thus, release a sample by Algorithm 3 preserves ε-differential privacy. It compromises the privacy guarantee to $(\varepsilon, (1+e^{\varepsilon})\delta)$ if the distribution (where the sample from) is δ-far away from the true posterior distribution, proved by Wang et al. [124]. $\square$

Note that when $\varepsilon = 4B$, the differentially private sampling process is identical to the non-private sampling. This is also the meaning of *some extent of free privacy*. It starts to lose accuracy when $\varepsilon < 4B$. One concern of this sampling approach is the distance δ between the distribution where the samples from and the true posterior distribution, which compromises the differential privacy guarantee. Fortunately, an emerging line of works, such as [105, 121], proved that SGLD can converge in finite iterations. As such we can have arbitrarily small δ with a (large) number of iterations.

3.5 Evaluation and Comparison

We test the proposed solutions on two real world datasets, ML100K and ML1M [81], which are widely employed for evaluating recommender systems. ML100K dataset has 100K ratings that 943 users assigned to 1682 movies. ML1M dataset contains 1 million ratings that 6040 users gave to 3952 movies. In the experiments, we adopt 5-fold cross validation for training and evaluation. We use root mean square error (RMSE) to measure accuracy performance:

$$RMSE = \sqrt{\frac{\sum_{(u,i)\in R^T}(r_{ui} - \hat{r}_{ui})^2}{|R^T|}}$$

where $|R^T|$ is the total number of ratings in the test set R^T. The lower the RMSE value the higher the accuracy. As a result of cross validation, the RMSE value reported in the following figures is the mean value of multiple runs.

3.5.1 Experiments Setup

In the following, the differentially-private SGD based PNBM is referred to as DPSGD-PNBM, and the differentially-private posterior sampling PNBM is referred as DPPS-PNBM. The experiment source code is available at Github[1].

We compare their performances with the following (state-of-the-art) baseline algorithms.

- *non-private PCC and COS:* There exist differentially-private NBMs based on Pearson correlation (PCC) or Cosine similarity (COS) NBMs (e.g. [46, 76, 135]). Since their accuracy is worse than the non-private algorithms, we directly focus on these non-private ones.

- *DPSGD-MF:* Differentially private matrix factorization from [9], which calibrates Laplacian noise into the SGD training process.

- *DPPS-MF:* Differentially private matrix factorization from [73], which exploits the posterior sampling technique.

We empirically choose the optimal parameters for each model using a heuristic grid search method. We summarize them as follows.

[1]https://github.com/lux-jwang/Experiments/tree/master/dpnbm

- *DPSGD-PNBM:* The learning rate η is searched in $\{0.1, 0.4\}$, and the iteration number $K \in [1, 20]$, the regular parameter $\lambda \in \{0.05, 0.005\}$, the rescale parameter $\beta \in \{10, 20\}$. The neighbor size $N_k = 500$, the lower bound of $|S_i|I_u^- : C \in \{10, 15\}$. In the training process, we decrease K and increase $\{\eta, C\}$ when requiring a stronger privacy guarantee (a smaller ε).

- *DPPS-PNBM:* The initial learning rate $\eta_1 \in \{8 \cdot 10^{-8}, 4 \cdot 10^{-7}, 8 \cdot 10^{-6}\}$, $\lambda \in \{0.02, 0.002\}$, the temperature parameter $\varrho = \{0.001, 0.006, 0.09\}$, the decay parameter $\xi = 0.3$. $N_k = 500$.

- *DPSGD-MF:* $\eta \in \{6 \cdot 10^{-4}, 8 \cdot 10^{-4}\}$, $K \in [10, 50]$ (the smaller privacy loss ε the less iterations), $\lambda \in \{0.2, 0.02\}$, the latent feature dimension $d \in \{10, 15, 20\}$.

- *DPPS-MF:* $\eta \in \{2{\cdot}10^{-9}, 2{\cdot}10^{-8}, 8{\cdot}10^{-7}, 8{\cdot}10^{-6}\}, \lambda \in \{0.02, 0.05, 0.1, 0.2\}$, $\varrho = \{1 \cdot 10^{-4}, 6 \cdot 10^{-4}, 4 \cdot 10^{-3}, 3 \cdot 10^{-2}\}$, $d \in \{10, 15, 20\}$,$\xi = 0.3$.

- *non-private PCC and COS:* For ML100K, we set $N_K = 900$. For ML1M, we set $N_K = 1300$.

3.5.2 Comparison Results

We first compare the accuracy between DPSGD-PNBM, DPSGD-MF, non-private PCC and COS and show the results in Fig. 3.5.1 for the two datasets respectively. When $\varepsilon \geq 20$, DPSGD-MF does not lose much accuracy, and it is better than non-private PCC and COS. However, the accuracy drops quickly (or, the RMSE increase quickly) when the privacy loss ε is reduced. This matches the observation in [9]. In the contrast, DPSGD-PNBM maintains a promising accuracy when $\varepsilon \geq 1$, and is better than non-private PCC and COS.

DPPS-PNBM and DPPS-MF preserve differential privacy at user level. We denote the privacy loss ε in form of $x \times \tau$ where x is a float value which indicates the average privacy loss at a rating level, and τ is the max rate number per user. The

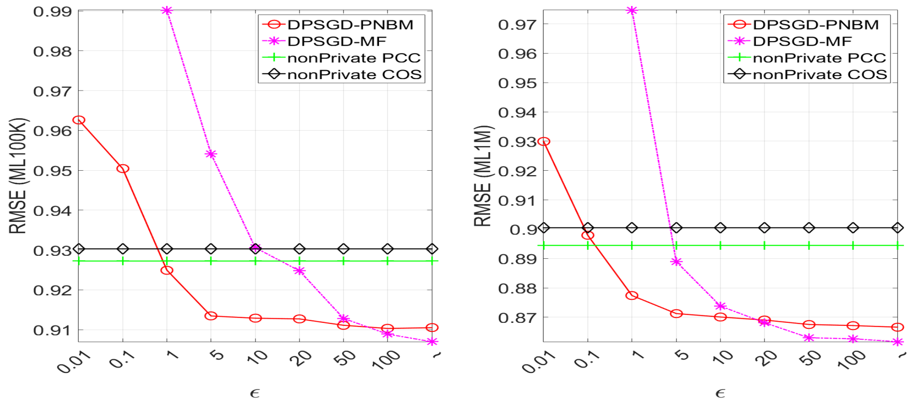


Figure 3.5.1: Accuracy Comparison: DPSGD-PNBM, DPSGD-MF, non-private PCC, COS.

comparison is shown in Fig. 3.5.2. In our context, for both datasets, $\tau = 200$. Both DPPS-PNBM and DPPS-MF allow accurate estimations when $\varepsilon \geq 0.1 \times 200$. It may seem that $\varepsilon = 20$ is a meaningless privacy guarantee. We remark that the average privacy of a rating level is 0.1. Besides the accuracy performance is better than the non-private PCC and COS, from the point of privacy loss ratio, our models match previous works [73, 76], where the authors showed that differentially private systems may not lose much accuracy when $\varepsilon > 1$.

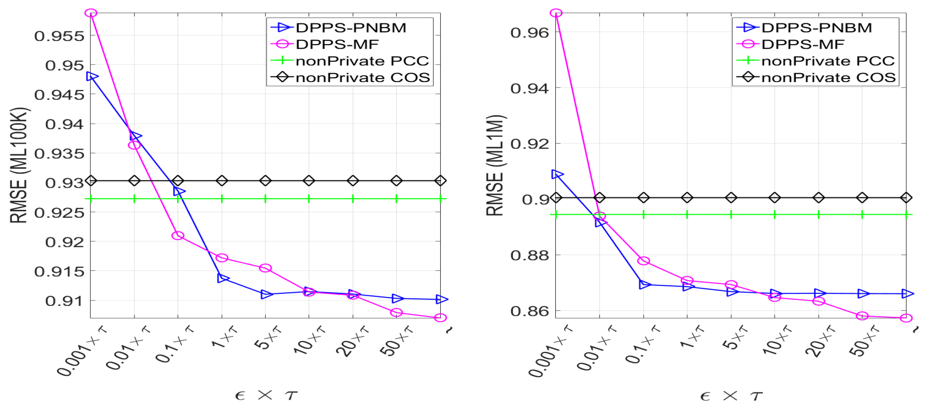


Figure 3.5.2: Accuracy Comparison: DPPS-PNBM, DPPS-MF, non-private PCC, COS.

For bandwidth and efficiency reason, mobile service providers may prefer to store the trained model (e.g. item *similarity*) in mobile devices directly. Commercial recommender systems often have very large *similarity* matrix such that the shortage of memory space in mobile devices may become a bottleneck. In order to alleviate this issue, we choose the *Top-N* most similar neighbors only by *similarity* matrix, by removing the rest neighbors of each item, such that we can sparsely store the matrix in practice. We compare accuracy with different number of neighbors with $\varepsilon = 1$, and summarize the results in Fig. 3.5.3. We stress two observations. Both DPSGD-PNBM and DPPS-PNBM reach their best accuracy with a smaller neighbor size. The accuracy of both DPSGD-PNBM and DPPS-PNBM is less sensitive than PCC and COS, when neighbor size is changed. This helps mitigate over-fitting problem and enhance system robustness.

DPSGD-PNBM and DPPS-PNBM achieve differential privacy at rating level (a single rating) and user level (a whole user profile) respectively. Below, we try to compare them at rating level, precisely at the average rating level for DPPS-PNBM. Fig. 3.5.4 shows that both solutions can obtain quite accurate predictions with a privacy guarantee ($\varepsilon \approx 1$). With the same privacy guarantee, DPPS-PNBM seems to be more accurate. However, DPPS-PNBM has its potential drawback. Recall

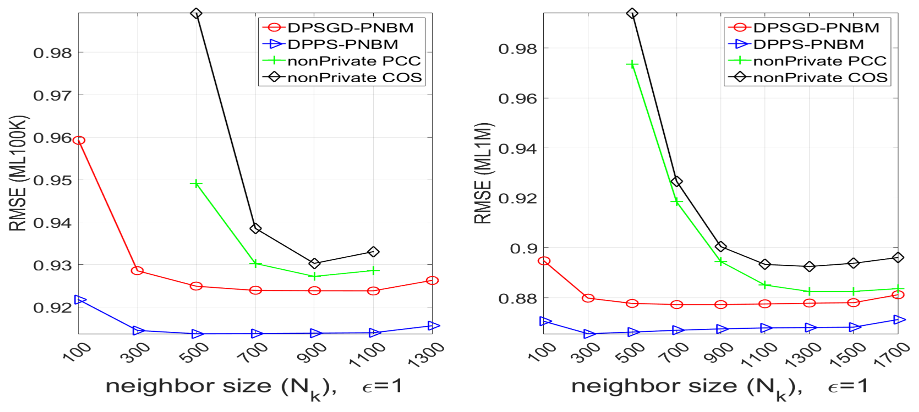


Figure 3.5.3: Accuracy comparison with different neighbor sizes

from Section 3.4, the difference δ between the distribution where samples from and the true posterior distribution compromises differential privacy guarantee. In order to have an arbitrarily small δ, DPPS-PNBM requires a large number of iterations [105, 121]. At this point, it is less efficient than DPSGD-PNBM. In our comparison, we assume $\delta \to 0$.

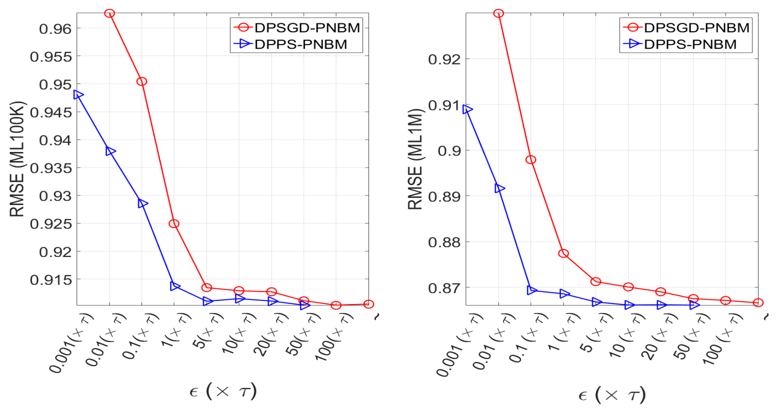


Figure 3.5.4: Accuracy comparison between DPSGD-PNBM and DPPS-PNBM

3.5.3 Summary

In summary, DPSGD-MF and DPPS-MF are more accurate when privacy loss is large (e.g. in a non-private case). DPSGD-PNBM and DPPS-PNBM are better when we want to reduce the privacy loss to a meaningful range. Both our models consistently outperform non-private traditional NBMs, with a meaningful differential privacy guarantee. Note that *similarity* is independent of NBM itself, thus other neighborhood-based recommenders can use our models to differential-privately learn *Similarity*, and deploy it to their existing systems without requiring extra effort.

3.6 Related Work

A number of works have demonstrated that an attacker can infer the user sensitive information, such as gender and politic view, from public recommendation results without using much background knowledge [16, 38, 82, 125].

Randomized data perturbation is one of earliest approaches to prevent user data from inference attack in which people either add random noise to their profiles or substitute some randomly chosen ratings with real ones (e.g. [96–98]). While this approach is very simple, it does not offer rigorous privacy guarantee. Differential privacy [33] aims to precisely protect user privacy in statistical databases, and the concept has become very popular recently. [76] is the first work to apply differential privacy to recommender systems, and it has considered both neighborhood-based methods (using correlation as *similarity*) and latent factor model (e.g. SVD). [135] introduced a differentially private neighbor selection scheme by injecting Laplace noise to the *similarity* matrix. [46] presented a scheme to obfuscate user profiles that preserves differential privacy. [9, 73] applied differential privacy to matrix factorization, and we have compared our solutions to theirs in Section 3.5.

3.7 Conclusion and Future Work

In this chapter, we have proposed two different differentially private NBMs, under a probabilistic framework. We firstly introduced a way to differential-privately find the maximum a posteriori *similarity* by calibrating noise to the SGD training process. Then we built differentially private NBM by exploiting the fact that sampling from scaled posterior distribution can result in differentially private systems. While the experiment results have demonstrated that our models allow promising accuracy with a modest privacy budget in some well-known datasets, we consider it as an interesting future work to test the performances in other real world datasets.

4 Social-context Facilitates Privacy-preserving Recommender Systems

4.1 Introduction

The accuracy of recommender systems relies on massive user history data, such as purchasing records, ratings, locations and so on. Leaking private data to others (including recommendation service providers) may arise privacy concerns to the public. Cryptographic primitives such as homomorphic encryption [42] and secure multiparty computation [71] are most commonly used tools to guarantee data security. However, the computational and spatial complexity of algebraic operations in a cipher space conflicts with the fact that accurate recommendations require a large number of mathematical operations (i.e., from massive training data and complicated learning processes). Taking two classes of representative recommendation algorithms as example,

- K-nearest neighborhood based [114]. Firstly, it computes similarities between any two users (resp. items); Secondly, it selects top-K the most similar users (resp. items) as the neighbors of a target user (resp. item). Lastly, it computes predictions based on the neighbors.

- Matrix factorization based [64]. Firstly, it decomposes a sparse rating matrix into two dense low-dimension feature matrices, where the feature metrics are iteratively learned by an optimization method such as stochastic gradient descent. Then it calculates predictions by performing dot-production over the learned features.

Obviously, executing either of them, in a cipher space, will result in a prohibitive cost of time and computational resource, especially, when considering another fact that accurate recommendation computations often rely on massive user data. Therefore, reducing the user data and simplifying the recommendation computation process can be a promising approach to improve the efficiency.

Our Contributions. We propose a social-context based recommender system which allows computing recommendations with only a few users without compromising the accuracy performance. We assume that the users have certain social connections. For convenience, we denote the social connection as friendship, i.e., any two users are friends if they have some kind of social connection, otherwise, we say the two users are strangers to each other. Note that we assume the friendship is public. In fact, in the real world, the friendship information are often public and can be easily discovered, such as using Facebook, Twitter and so on.

- We experimentally validate the assumption that friends often have similar interests, by the datasets we collected from Twitter. Besides serving for evaluating our recommender protocols, the datasets are also an independent interests for the community.

- We proposed a friendship-based recommender system based on a neighborhood method, which computes recommendations to a user with the data from a few of his/her friends and strangers. This friendship-based recommender system can significantly reduce the computational complexity while allowing a promising accuracy performance.

- Based on our friendship-based recommender system and a somewhat (fully) homomorphic encryption scheme, we construct two secure protocols, one for single value predictions and the other for Top-n recommendations. The security of protocol executions is straightforwardly guaranteed by the underlying homomorphic encryption scheme, and we further discuss the information leakage in algorithm outputs.

- We provide both asymptotic and implementation results for our proposed protocols. We show that the single prediction protocol is very efficient while the Top-n protocol is not. We further discuss two relaxations for the top-n protocol and show that they are quite efficient.

4.2 Assumption Validation

In this section, we present an intuitive evidence that (1) individuals are highly socially-connected, (2) individuals with social connections often have similar interests.

4.2.1 Dataset Construction

To validate our assumptions, we construct a new dataset containing social connection information, based on the dataset MovieTweetings[27]. MovieTweetings (MT) consists of ratings on movies that are extracted from tweets. Such tweets are originated from the social rating widget available in IMDb apps. We use a snapshot of the MT dataset which contains 359908 ratings, 35456 users and 20156 items. Note that in the MT dataset, each user has at least 1 rating, but without any social-connection information. We crawled the followees of each user ID recorded in the

MT dataset from Twitter. Based on the *"following"* activities in Twitter, we naturally introduce the concept of friendship as follows: if a user x follows user y then we say user x regards user y as a friend. Note that friendship is not guaranteed to be bi-directional, namely users x and y may not consider each other as friends at the same time. We name our new dataset as Friendship MovieTweetings (FMT). It is worth stressing that, in the new dataset, we only collect the Twitter users who have explicitly posted their movie ratings. In the other word, the friend list of a user may be incomplete. In our experiment, we only use a subset of FMT, in which each user has at least 10 friends and each friend has at least 10 ratings. The rating scale is regularized to [0,5]. This subset will be referred to as 10-FMT in this chapter. We summarize the basic information of these datasets in Table 4.2.1.

	MT	**FMT**	**10-FMT**
Ratings Num	359908	211954	20316
Users Num	35456	17268	508
Items Num	20156	15682	3481
Matrix Density	0.050%	0.078%	1.15%
Min Ratings/User	1	1	10
Ave. Ratings/User	10	12	39
Max Ratings/User	856	856	448
Min Friends/User	–	1	10
Ave. Friends/User	–	6	26
Max Friends/User	–	282	109

Table 4.2.1: Datasets

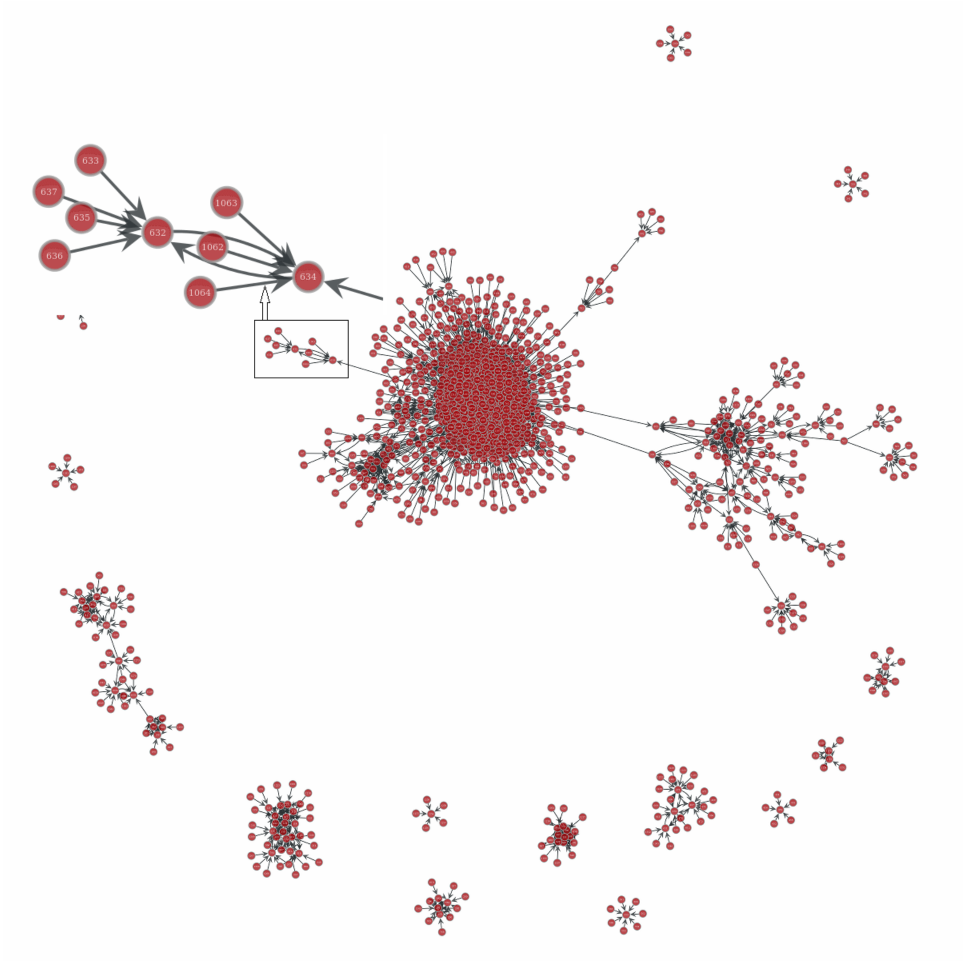


Figure 4.2.1: Social Graph of FMT Dataset

We map the FMT dataset into a directed graph in Figure 4.2.1. In the graph, a node represents a user. If there is a directed edge from user *x* to user *y*, then user *x* regards user *y* as a friend. It is clear that almost all users are connected in the social graph. To enlarge the number of friends of each user, we also count the number of friends of friends (FoFs), which are used in designing the decentralized protocol. The number of friends are summarized in Table 4.2.2. Counting the FOFs as a user's friends makes sense. For example, saying that user *z* is a friend of user *y*, and the user *y* is a friend of user *x*, when the user *z* posts a tweet which may be re-tweeted by the user y, the user x would see the tweet (i.e., be influenced by the user z directly).

	Min Friend Num	**Avg Friend Num**	**Max Friend Num**
Friends	10	27	109
FoFs	10	203	329

Table 4.2.2: Basic Facts

In order to test the friends often have similar interests, we compute the Cosine similarities between users in the 10-FMT dataset and plot them in Figure 4.2.2. We also calculate the similarities between strangers. As shown as in Figure 4.2.2, averagely, friends share more similar preferences than strangers. Therefore, the observations match our assumptions. We will show how to construct an accurate and efficient recommender system while preserving privacy, based on this fact.

4.3 Our Approach

To construct recommender systems based on collaborative filtering techniques [114], there two typical approaches, memory-based approach such as neighborhood-based methods [25]; model-based approach such as matrix factorization [64]. Our setting is constrained to the following facts,

- The number of friends who agreed to contribute may be very small.
- A lower computational complexity is always more preferable.

We build our recommendation algorithm on a neighborhood-based method. In contrast, matrix factorization always requires iteratively training the model with massive user data. On the one hand, it leads to a much higher computational

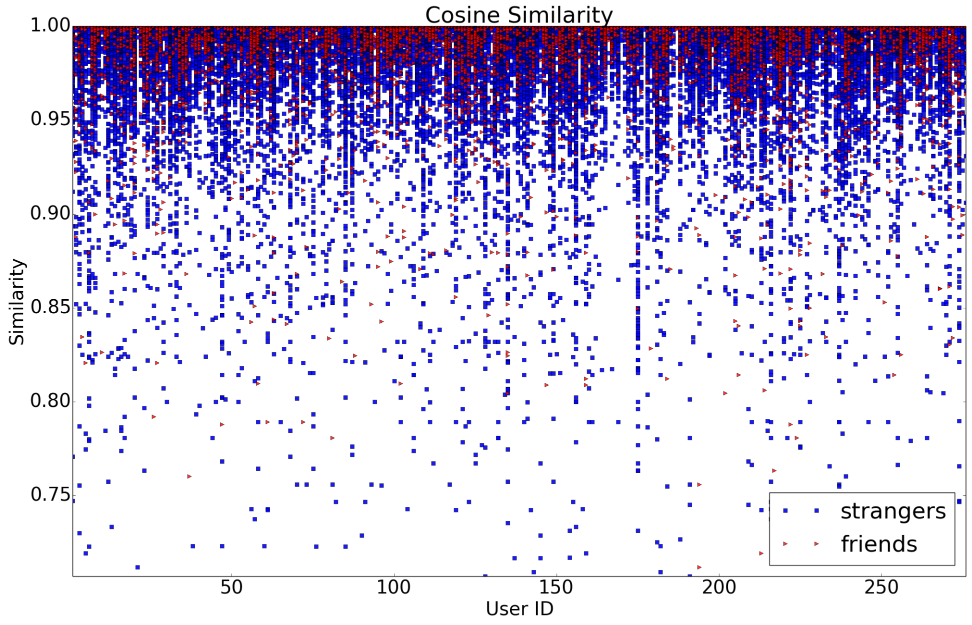


Figure 4.2.2: Cosine Similarity of 10-FMT Dataset

complexity; on the other hand, a very limited number of friends are not sufficient enough to train an accurate factorization model.

4.3.1 Our Recommendation Model

In our solution, we compute the predicted rating for user u based on inputs from both his friends $\mathbf{F}_u$ and some strangers $\mathbf{T}_u$ for both accuracy and security reasons. Though friends often have common preferences and more impacts to each other, in reality, it is common that the number of friends who consumed a target item can be few. If recommendations are computed solely based on the inputs of user u's friends, the accuracy performance can be compromised, and the private information of user u's friends might be leaked through user u's outputs. Therefore, it is reasonable to believe that, by taking into account some randomly chosen strangers, we will mitigate both problems.

Given an active user u, when factoring in the inputs from randomly chosen strangers, we will use the simple Bias From Mean (BFM) scheme for the purpose of simplicity. It is worth stressing that there are a lot of different choices for this task. Nevertheless, as to the accuracy, this scheme has similar performance to many other more sophisticated schemes, such as Slope One and Pearson/Cosine similarity-based collaborative filtering schemes [67]. Let the stranger set be $\mathbf{T}_u$, the predicted value $p^{*}_{u,b}$ for item b is computed as follows.

$$p^{*}_{u,b} = \overline{r_u} + \frac{\sum_{t \in \mathbf{T}_u} q_{t,b} \cdot (r_{t,b} - \overline{r_t})}{\sum_{t \in \mathbf{T}_u} q_{t,b}} \tag{4.1}$$

where $q_{t,b}$ is a binary indicator, where $q_{t,b} = 1$ if a stranger t rated item b, otherwise, $q_{t,b} = 0$. When factoring in the inputs from the friends, we let the friend set be $\mathbf{F}_u$, the predicted value $p^{**}_{u,b}$ for item b is computed as follows.

$$p^{**}_{u,b} = \overline{r_u} + \frac{\sum_{f \in \mathbf{F}_u} q_{f,b} \cdot (r_{f,b} - \overline{r_f}) \cdot w_{u,f}}{\sum_{f \in \mathbf{F}_u} q_{f,b} \cdot w_{u,f}} \tag{4.2}$$

where $q_{f,b}$ is a binary indicator, where $q_{f,b} = 1$ if a friend f rated item b, otherwise, $q_{f,b} = 0$. In practice, the similarity between friends means that they tend to prefer similar items. However, this does not imply that they will assign very similar scores to the items. For example, a user Alice may be very mean and assign a score 3 to most of her favorite items while her friends may be very generous and assign a score 5 to their favorite items. Using the Equation (1), we will likely generate a score 5 for an unrated item for Alice, who may just rate a score 3 for the item even if she likes it. In this regard, Equation (4.2) is more appropriate because $\overline{r_u}$ reflects the user's rating style and $\frac{\sum_{f \in \mathbf{F}_u} q_{f,b} \cdot (r_{f,b} - \overline{r_f}) \cdot w_{f,u}}{\sum_{f \in \mathbf{F}_u} q_{f,b} \cdot w_{f,u}}$ reflects the user's preference based on inputs from his friends.

Based on the inputs from the strangers and friends, a combined predicted value $p_{u,b}$ for an unrated item b can be computed as $p_{u,b} = \rho \cdot p^{*}_{u,b} + (1 - \rho) \cdot p^{**}_{u,b}$ for some $0 \leq \rho \leq 1$. Due to the fact that cryptographic primitives are often designed for dealing with integers, we rephrase the formula as follows, where α, β are integers.

$$p_{u,b} = \frac{\beta}{\alpha + \beta} \cdot p^{*}_{u,b} + \frac{\alpha}{\alpha + \beta} \cdot p^{**}_{u,b} \tag{4.3}$$

It is worth noting that the prediction $p_{u,b}$ is not deterministic because we assume the stranger set $\mathbf{T}_u$ is randomly chosen for the computation.

4.3.2 Threat Model

As to communication, we assume all communications are mediated by a recommender system (RS) server and the communication channels are integrity and confidentiality protected. Instead of making a general semi-honest assumption on all participants, we distinguish the following.

- Threat from semi-honest RS server. In the view of all users, the RS server will follow the protocol specification but it may try to infer their private information from openly collected transaction records.

- Threat from a semi-honest friend. In the view of a user, none of his friends will collude with the RS server or another party to breach his privacy. We believe the social norm deters such colluding attacks, and the deterrence comes from the fact that once such a collusion is known to the victim user then the friendship may be jeopardized. Nevertheless, we still need to consider possible privacy threats in two scenarios.

  - In the view of $f \in \mathbf{F}_u$, user u may attempt to learn his private information when running the recommendation protocol. In the view of user u, his friend $f \in \mathbf{F}_u$ may also try to infer his information as well.
  - In the view of $f \in \mathbf{F}_u$, user u's output (e.g. a new rated item and predicted rating value) may be leaked. If another party obtains such auxiliary information, then user f's private information may be at risk. For example, the Potential Information Leakage through Friends security issue in Section 3.1 falls into this scenario.

- Threat from strangers. We consider the following two scenarios.
  - In the view of user u and his friends, a stranger may try to learn their private information.
  - In the view of a stranger $t \in \mathbf{T}_u$, who is involved in the protocol execution of user u, user u may try to learn his private information.

4.4 Centralized Friendship-based Protocols

We generally assume that there is a recommender service provider, which will maintain the social graph and mediate the executions of recommender protocols among users. The system structure is shown in Figure. 4.4.1.

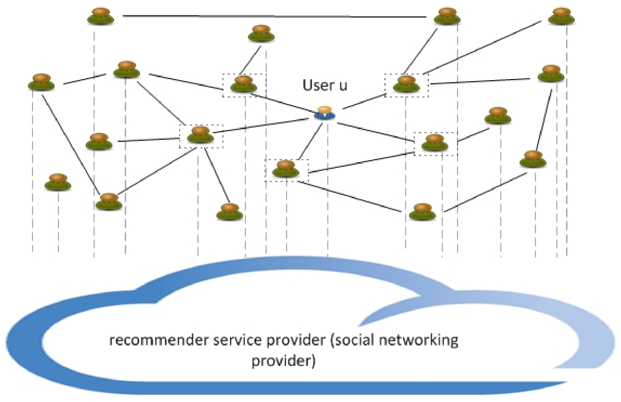


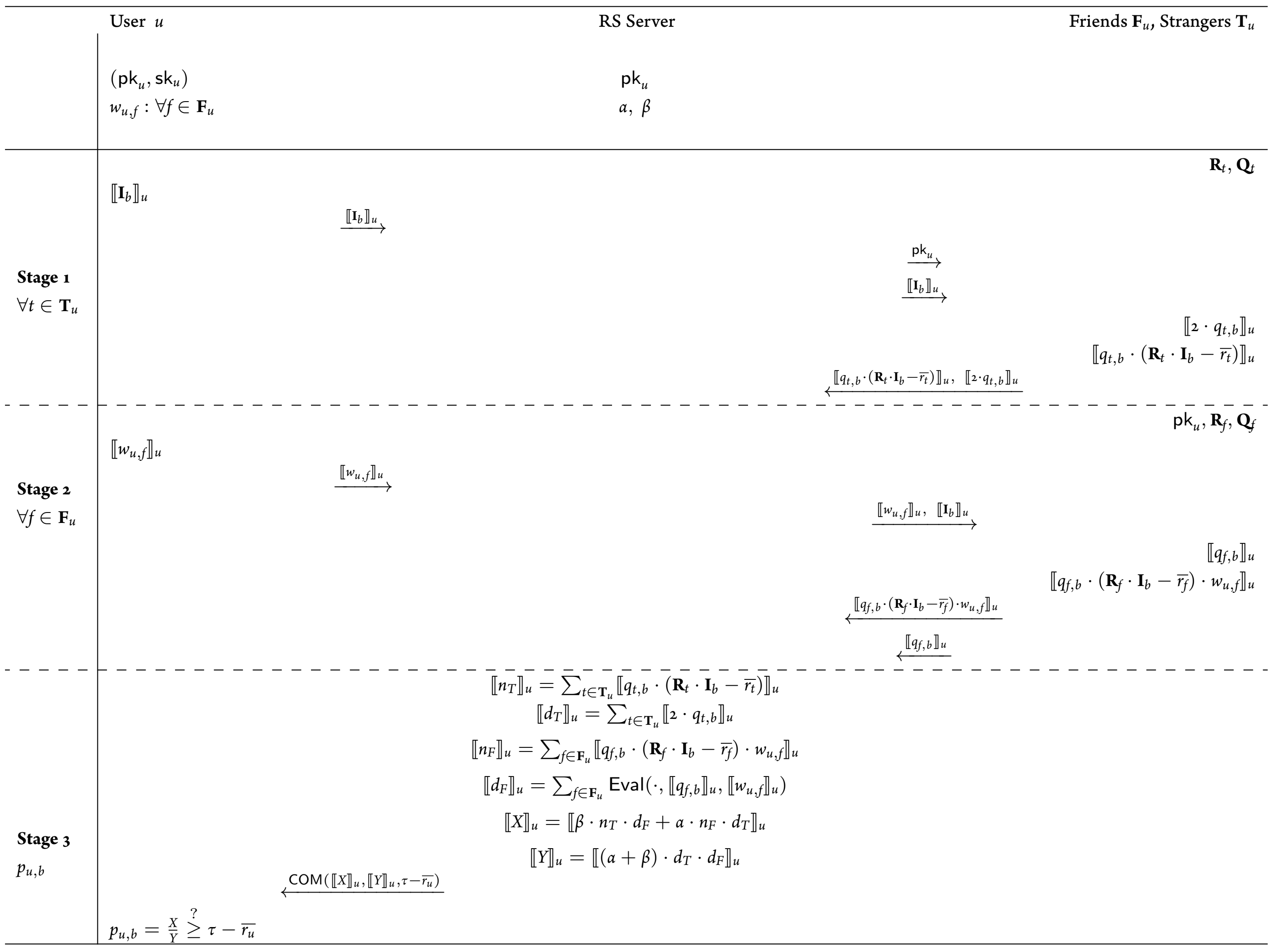
Figure 4.4.1: System Structure in the View of User u

With respect to the tailored recommender algorithms in Section 4.3, the global system parameters should be established in advance. Such parameters should include α, β which determine how a predicted rating value for user u is generated based on the inputs of friends and strangers, and they should also include the size of stranger set $\mathbf{T}_u$. In the initialization phase, user u generates his public/private key pair $(\mathsf{pk}_u, \mathsf{sk}_u)$ for a somewhat (fully) homomorphic encryption (SWHE) scheme and sends pk_u to the server. We require that the SWHE scheme allows to encrypt negative integers. In addition, user u maintains a rating vector $\mathbf{R}_u$, his social graph, and assigns a weight $w_{u,f}$ to each of his friend $f \in \mathbf{F}_u$. All other users perform the same operations in this phase.

Under the centralized setting, we describe two secure protocols, one is for single value prediction (Section 4.4.1), the other is for Top-n recommendation (Section 4.4.2).

4.4.1 Centralized Single Prediction Protocol

When user u wants to test whether the predicted rating for an unrated item b is above a certain threshold τ (an integer) in his mind, he initiates the protocol in Figure. 4.4.2. Referring to the prediction algorithm from Section 4.3.1, in stage 1 the service provider collects the inputs from the strangers in encrypted form according to Equation (4.1), while in stage 2 the service provider collects the inputs from the friends in encrypted form according to Equation (4.2). In stage 3, user u learns whether the prediction is above a threshold while the service provider learns nothing. In more detail, the protocol runs in three stages.



User u
RS Server
Friends $\mathbf{F}_u$, Strangers $\mathbf{T}_u$
$(\mathsf{pk}_u, \mathsf{sk}_u)$
$w_{u,f} : \forall f \in \mathbf{F}_u$
pk_u
$\alpha,\ \beta$
$\mathbf{R}_t, \mathbf{Q}_t$
$[\![\mathbf{I}_b]\!]_u$
$\xrightarrow{[\![\mathbf{I}_b]\!]_u}$
Stage 1
$\forall t \in \mathbf{T}_u$
$\xrightarrow{\mathsf{pk}_u}$
$\xrightarrow{[\![\mathbf{I}_b]\!]_u}$
$[\![2 \cdot q_{t,b}]\!]_u$
$[\![q_{t,b} \cdot (\mathbf{R}_t \cdot \mathbf{I}_b - \overline{r_t})]\!]_u$
$\xleftarrow{[\![q_{t,b} \cdot (\mathbf{R}_t \cdot \mathbf{I}_b - \overline{r_t})]\!]_u,\ [\![2 \cdot q_{t,b}]\!]_u}$
$\mathsf{pk}_u, \mathbf{R}_f, \mathbf{Q}_f$
$[\![w_{u,f}]\!]_u$
$\xrightarrow{[\![w_{u,f}]\!]_u}$
Stage 2
$\forall f \in \mathbf{F}_u$
$\xrightarrow{[\![w_{u,f}]\!]_u,\ [\![\mathbf{I}_b]\!]_u}$
$[\![q_{f,b}]\!]_u$
$[\![q_{f,b} \cdot (\mathbf{R}_f \cdot \mathbf{I}_b - \overline{r_f}) \cdot w_{u,f}]\!]_u$
$\xleftarrow{[\![q_{f,b} \cdot (\mathbf{R}_f \cdot \mathbf{I}_b - \overline{r_f}) \cdot w_{u,f}]\!]_u}$
$\xleftarrow{[\![q_{f,b}]\!]_u}$
$[\![n_T]\!]_u = \sum_{t \in \mathbf{T}_u} [\![q_{t,b} \cdot (\mathbf{R}_t \cdot \mathbf{I}_b - \overline{r_t})]\!]_u$
$[\![d_T]\!]_u = \sum_{t \in \mathbf{T}_u} [\![2 \cdot q_{t,b}]\!]_u$
$[\![n_F]\!]_u = \sum_{f \in \mathbf{F}_u} [\![q_{f,b} \cdot (\mathbf{R}_f \cdot \mathbf{I}_b - \overline{r_f}) \cdot w_{u,f}]\!]_u$
$[\![d_F]\!]_u = \sum_{f \in \mathbf{F}_u} \mathsf{Eval}(\cdot, [\![q_{f,b}]\!]_u, [\![w_{u,f}]\!]_u)$
$[\![X]\!]_u = [\![\beta \cdot n_T \cdot d_F + \alpha \cdot n_F \cdot d_T]\!]_u$
$[\![Y]\!]_u = [\![(\alpha + \beta) \cdot d_T \cdot d_F]\!]_u$
Stage 3
$p_{u,b}$
$\xleftarrow{\mathsf{COM}([\![X]\!]_u, [\![Y]\!]_u, \tau - \overline{r_u})}$
$p_{u,b} = \frac{X}{Y} \overset{?}{\geq} \tau - \overline{r_u}$

Figure 4.4.2: Single Prediction Protocol

1. In the first stage, the participants interact as follows.

   (a) User u generates a binary vector $\mathbf{I}_b$, which only has 1 for the b-th element, and sends the ciphertext $[\![\mathbf{I}_b]\!]_u = \mathsf{Enc}(\mathbf{I}_b, \mathsf{pk}_u)$ to the server.

Let's assume $[\![\mathbf{I}_b]\!]_u = ([\![\mathbf{I}_b^{(1)}]\!]_u, \cdots, [\![\mathbf{I}_b^{(M)}]\!]_u)$.

(b) The server first sends pk_u to some randomly chosen strangers, and see whether they want to participate in the computation.

(c) After the server has successfully found a viable stranger set $\mathbf{T}_u$, it forwards $[\![\mathbf{I}_b]\!]_u$ to every user in $\mathbf{T}_u$.

(d) With pk_u and $(\mathbf{R}_t, \mathbf{Q}_t)$, every user t from $\mathbf{T}_u$ can compute the following based on the homomorphic properties.

$$[\![2 \cdot q_{t,b}]\!]_u = \sum_{1 \le i \le M} \mathsf{Eval}(\cdot, \mathsf{Enc}(\mathsf{pk}_u, 2 \cdot q_{t,i}), [\![\mathbf{I}_b^{(i)}]\!]_u)$$

$$[\![\mathbf{R}_t \cdot \mathbf{I}_b]\!]_u = \sum_{1 \le i \le M} \mathsf{Eval}(\cdot, \mathsf{Enc}(r_{t,i}, \mathsf{pk}_u), [\![\mathbf{I}_b^{(i)}]\!]_u)$$

$$\begin{aligned} &[\![q_{t,b} \cdot (\mathbf{R}_t \cdot \mathbf{I}_b - \overline{r_t})]\!]_u \\ =&\mathsf{Eval}(\cdot, [\![q_{t,b}]\!]_u, \mathsf{Eval}(+, [\![\mathbf{R}_t \cdot \mathbf{I}_b]\!]_u, \mathsf{Enc}(-\overline{r_t}, \mathsf{pk}_u))) \end{aligned}$$

2. In the second stage, the participants interact as follows.

(a) For every friend $f \in \mathbf{F}_u$, user u sends the encrypted weight $[\![w_{u,f}]\!]_u = \mathsf{Enc}(w_{u,f}, \mathsf{pk}_u)$ to the server.

(b) The server sends $[\![w_{u,f}]\!]_u$ and $[\![\mathbf{I}_b]\!]_u$ to user f.

(c) With pk_u, $[\![\mathbf{I}_b]\!]_u$, $[\![w_{u,f}]\!]_u$ and $(\mathbf{R}_f, \mathbf{Q}_f)$, user f can compute the following.

$$[\![q_{f,b}]\!]_u = \sum_{1 \le i \le M} \mathsf{Eval}(\cdot, \mathsf{Enc}(q_{f,i,\mathsf{pk}_u}), [\![\mathbf{I}_b^{(i)}]\!]_u)$$

$$[\![\mathbf{R}_f \cdot \mathbf{I}_b]\!]_u = \sum_{1 \le i \le M} \mathsf{Eval}(\cdot, \mathsf{Enc}(r_{f,i}, \mathsf{pk}_u), [\![\mathbf{I}_b^{(i)}]\!]_u)$$

$$
\begin{aligned}
[\![q_{f,b} \cdot (\mathbf{R}_f \cdot \mathbf{I}_b - \overline{r_f}) \cdot w_{u,f}]\!]_u = & \mathsf{Eval}(\cdot, \mathsf{Eval}(\cdot, [\![q_{f,b}]\!]_u, [\![w_{u,f}]\!]_u), \\
& \mathsf{Eval}(+, [\![\mathbf{R}_f \cdot \mathbf{I}_b]\!]_u, \mathsf{Enc}(-\overline{r_f}, \mathsf{pk}_u)))
\end{aligned}
$$

3. In the third stage, user u and the server interact as follows.

   (a) The server first computes $[\![n_T]\!]_u$, $[\![d_T]\!]_u$, $[\![n_F]\!]_u$, $[\![d_F]\!]_u$ as shown in Figure. 4.4.2, and then computes $[\![X]\!]_u$, $[\![Y]\!]_u$ as follows.

$$
\begin{aligned}
temp_1 &= \mathsf{Eval}(\cdot, \mathsf{Eval}(\cdot, [\![n_T]\!]_u, [\![d_F]\!]_u), \mathsf{Enc}(\beta, \mathsf{pk}_u)) \\
temp_2 &= \mathsf{Eval}(\cdot, \mathsf{Eval}(\cdot, [\![n_F]\!]_u, [\![d_T]\!]_u), \mathsf{Enc}(\alpha, \mathsf{pk}_u)) \\
[\![X]\!]_u &= \mathsf{Eval}(+, temp_1, temp_2) \\
[\![Y]\!]_u &= \mathsf{Eval}(\cdot, \mathsf{Eval}(\cdot, [\![d_F]\!]_u, [\![d_T]\!]_u), \mathsf{Enc}(\alpha + \beta, \mathsf{pk}_u))
\end{aligned}
$$

   Referring to Equations (4.1) and (4.2), we have $p^*_{u,b} = \overline{r_u} + \frac{n_T}{d_T}$ and $p^{**}_{u,b} = \overline{r_u} + \frac{n_F}{d_F}$. The ultimate prediction $p_{u,b}$ can be denoted as follows.

$$
\begin{aligned}
p_{u,b} &= \frac{\beta}{\alpha + \beta} \cdot p^*_{u,b} + \frac{\alpha}{\alpha + \beta} \cdot p^{**}_{u,b} \\
&= \overline{r_u} + \frac{\beta \cdot n_T \cdot d_F + \alpha \cdot n_F \cdot d_T}{(\alpha + \beta) \cdot d_T \cdot d_F} \\
&= \overline{r_u} + \frac{X}{Y}
\end{aligned}
$$

   (b) User u runs a comparison protocol COM with the server to learn whether $\frac{X}{Y} \geq \tau - \overline{r_u}$. Since $X, Y, \tau - \overline{r_u}$ are integers, COM is indeed an encrypted integer comparison protocol: where user u holds the private key sk_u and τ, the server holds $[\![X]\!]_u$, $[\![Y]\!]_u$, and the protocol outputs a bit to user u indicating whether $X \geq (\tau - \overline{r_u}) \cdot Y$.

4.4.2 Centralized Top-n Protocol

When the active user u wants to figure out Top-n unrated items, he initiates the protocol in Figure. 4.4.3. This protocol shares the same design philosophy as that of single prediction protocol. The usage of matrix $\mathbf{M}_X$ in the random permutation of Stage 3 guarantees that the rated items will all appear in the end of the list after ranking. As a result, the rated items will not appear in the recommended Top-n items. In more detail, the protocol runs in three stages.

1. In the first stage, the participants interact as follows.

   (a) The server sends pk_u to some randomly chosen strangers and see whether they want to participate in the computation. Suppose that the server has successfully found $\mathbf{T}_u$.

   (b) With pk_u and $(\mathbf{R}_t, \mathbf{Q}_t)$, user $t \in \mathbf{T}_u$ can compute $[\![q_{t,b} \cdot (r_{t,b} - \overline{r_t})]\!]_u = \mathsf{Enc}(q_{t,b} \cdot (r_{t,b} - \overline{r_t}), \mathsf{pk}_u)$ and $[\![2 \cdot q_{t,b}]\!]_u = \mathsf{Enc}(2 \cdot q_{t,b}, \mathsf{pk}_u)$ for every $1 \leq b \leq M$. All encrypted values are sent back to the server.

2. In the second stage, the participants interact as follows.

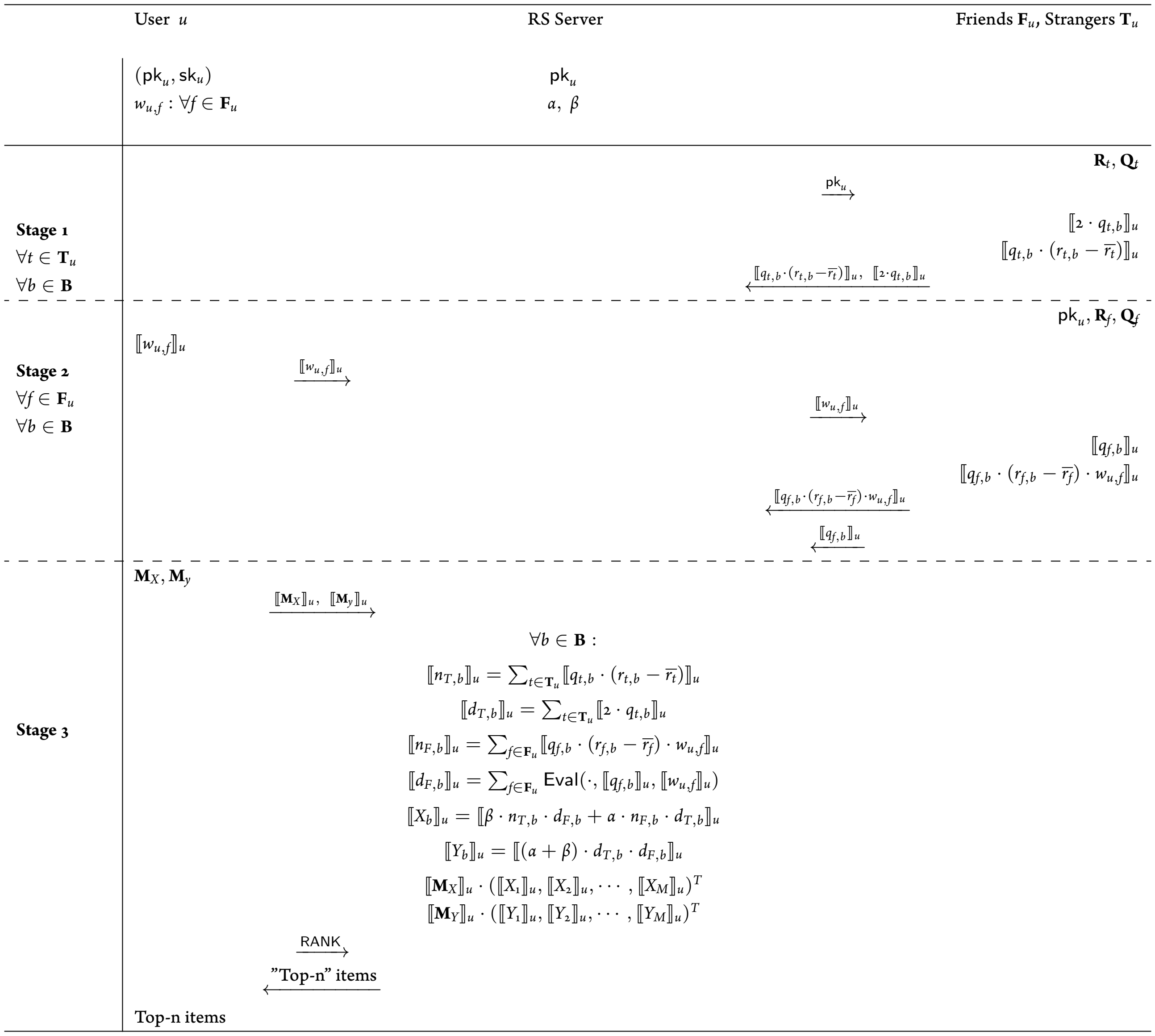

User u
RS Server
Friends $\mathbf{F}_u$, Strangers $\mathbf{T}_u$
$(\mathsf{pk}_u, \mathsf{sk}_u)$
$w_{u,f} : \forall f \in \mathbf{F}_u$
pk_u
$\alpha,\ \beta$
$\mathbf{R}_t, \mathbf{Q}_t$
$\xrightarrow{\mathsf{pk}_u}$
Stage 1
$\forall t \in \mathbf{T}_u$
$\forall b \in \mathbf{B}$
$[\![2 \cdot q_{t,b}]\!]_u$
$[\![q_{t,b} \cdot (r_{t,b} - \overline{r_t})]\!]_u$
$\xleftarrow{[\![q_{t,b} \cdot (r_{t,b} - \overline{r_t})]\!]_u,\ [\![2 \cdot q_{t,b}]\!]_u}$
$\mathsf{pk}_u, \mathbf{R}_f, \mathbf{Q}_f$
$[\![w_{u,f}]\!]_u$
Stage 2
$\forall f \in \mathbf{F}_u$
$\forall b \in \mathbf{B}$
$\xrightarrow{[\![w_{u,f}]\!]_u}$
$\xrightarrow{[\![w_{u,f}]\!]_u}$
$[\![q_{f,b}]\!]_u$
$[\![q_{f,b} \cdot (r_{f,b} - \overline{r_f}) \cdot w_{u,f}]\!]_u$
$\xleftarrow{[\![q_{f,b} \cdot (r_{f,b} - \overline{r_f}) \cdot w_{u,f}]\!]_u}$
$\xleftarrow{[\![q_{f,b}]\!]_u}$
$\mathbf{M}_X, \mathbf{M}_y$
$\xrightarrow{[\![\mathbf{M}_X]\!]_u,\ [\![\mathbf{M}_y]\!]_u}$
Stage 3
$\forall b \in \mathbf{B} :$
$[\![n_{T,b}]\!]_u = \sum_{t \in \mathbf{T}_u} [\![q_{t,b} \cdot (r_{t,b} - \overline{r_t})]\!]_u$
$[\![d_{T,b}]\!]_u = \sum_{t \in \mathbf{T}_u} [\![2 \cdot q_{t,b}]\!]_u$
$[\![n_{F,b}]\!]_u = \sum_{f \in \mathbf{F}_u} [\![q_{f,b} \cdot (r_{f,b} - \overline{r_f}) \cdot w_{u,f}]\!]_u$
$[\![d_{F,b}]\!]_u = \sum_{f \in \mathbf{F}_u} \mathsf{Eval}(\cdot, [\![q_{f,b}]\!]_u, [\![w_{u,f}]\!]_u)$
$[\![X_b]\!]_u = [\![\beta \cdot n_{T,b} \cdot d_{F,b} + \alpha \cdot n_{F,b} \cdot d_{T,b}]\!]_u$
$[\![Y_b]\!]_u = [\![(\alpha + \beta) \cdot d_{T,b} \cdot d_{F,b}]\!]_u$
$[\![\mathbf{M}_X]\!]_u \cdot ([\![X_1]\!]_u, [\![X_2]\!]_u, \cdots, [\![X_M]\!]_u)^T$
$[\![\mathbf{M}_Y]\!]_u \cdot ([\![Y_1]\!]_u, [\![Y_2]\!]_u, \cdots, [\![Y_M]\!]_u)^T$
$\xrightarrow{\mathsf{RANK}}$
$\xleftarrow{\text{"Top-n" items}}$
Top-n items

Figure 4.4.3: Top-n Protocol

(a) To every friend $f \in \mathbf{F}_u$, user u sends the encrypted weight $[\![w_{u,f}]\!]_u =$

$\mathsf{Enc}(w_{u,f}, \mathsf{pk}_u)$.

(b) With pk_u, $[\![w_{u,f}]\!]_u$ and $(\mathbf{R}_f, \mathbf{Q}_f)$, user f can compute $[\![q_{f,b}]\!]_u$ and

$$\begin{aligned}&[\![q_{f,b} \cdot (r_{f,b} - \overline{r_f}) \cdot w_{u,f}]\!]_u\\ =&\mathsf{Eval}(\cdot, \mathsf{Enc}(q_{f,b} \cdot (r_{f,b} - \overline{r_f}), \mathsf{pk}_u), [\![w_{u,f}]\!]_u)\end{aligned}$$

for every $1 \leq b \leq M$. All encrypted values are sent back to the server.

3. In the third stage, user u and the server interact as follows.

   (a) User u generates two matrices $\mathbf{M}_X, \mathbf{M}_Y$ as follows: (1) generate a $M \times M$ identity matrix; (2) randomly permute the columns to obtain $\mathbf{M}_Y$; (3) to obtain $\mathbf{M}_X$, for every b, if item b has been rated then replace the element 1 in b-th column with 0.

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \cdots & \cdots\cdots & \cdots & \\ 0 & 0 & \cdots & 1 \end{bmatrix} \rightarrow \mathbf{M}_Y = \begin{bmatrix} 0 & 1 & \cdots & 0 \\ 0 & 0 & \cdots & 1 \\ \cdots & \cdots\cdots & \cdots & \\ 1 & 0 & \cdots & 0 \end{bmatrix}$$

$$\rightarrow \mathbf{M}_X = \begin{bmatrix} 0 & 1 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \cdots & \cdots\cdots & \cdots & \\ 1 & 0 & \cdots & 0 \end{bmatrix}$$

   User u encrypts the matrices (element by element) and sends $[\![\mathbf{M}_X]\!]_u$, $[\![\mathbf{M}_Y]\!]_u$ to the server, which then proceeds as follows.

      i. The server first computes $[\![n_{T,b}]\!]_u$, $[\![d_{T,b}]\!]_u$, $[\![n_{F,b}]\!]_u$, $[\![d_{F,b}]\!]_u$, $[\![X_b]\!]_u$, $[\![Y_b]\!]_u$ for every $1 \leq b \leq M$ as shown in Figure. 4.4.3, in the same way as in the previous protocol in Figure. 4.4.2. Referring to Formula (4.3), we see that $\overline{r_u}$ appears in $p_{u,b}$ for every b. *For simplicity, we ignore this term when comparing the predictions for different unrated items.* With this simplification, the prediction

$p_{u,b}$ can be denoted as follows.

$$\begin{aligned} p_{u,b} &= \frac{\beta}{\alpha+\beta} \cdot \frac{n_{T,b}}{d_{T,b}} + \frac{\alpha}{\alpha+\beta} \cdot \frac{n_{F,b}}{d_{F,b}} \\ &= \frac{\beta \cdot n_{T,b} \cdot d_{F,b} + \alpha \cdot n_{F,b} \cdot d_{T,b}}{(\alpha+\beta) \cdot d_{T,b} \cdot d_{F,b}} \\ &= \frac{X_b}{Y_b} \end{aligned}$$

ii. The server permutes the ciphertexts vector $((\llbracket X_1 \rrbracket_u, \llbracket Y_1 \rrbracket_u), (\llbracket X_2 \rrbracket_u, \llbracket Y_2 \rrbracket_u), \cdots, (\llbracket X_M \rrbracket_u, \llbracket Y_M \rrbracket_u))$ in an oblivious manner as follows.

$$\begin{aligned} &(\llbracket U_1 \rrbracket_u, \llbracket U_2 \rrbracket_u, \cdots, \llbracket U_M \rrbracket_u) \\ =&\llbracket \mathbf{M}_X \rrbracket_u \cdot (\llbracket X_1 \rrbracket_u, \llbracket X_2 \rrbracket_u, \cdots, \llbracket X_M \rrbracket_u)^T \end{aligned}$$

$$\begin{aligned} &(\llbracket V_1 \rrbracket_u, \llbracket V_2 \rrbracket_u, \cdots, \llbracket V_M \rrbracket_u) \\ =&\llbracket \mathbf{M}_Y \rrbracket_u \cdot (\llbracket Y_1 \rrbracket_u, \llbracket Y_2 \rrbracket_u, \cdots, \llbracket Y_M \rrbracket_u)^T \end{aligned}$$

The multiplication between the ciphertext matrix and ciphertext vector is done in the standard way, except that the multiplication between two elements is done with $\mathsf{Eval}(\cdot, ,)$ and the addition is done with $\mathsf{Eval}(+, ,)$. Suppose item b has been rated before and $(\llbracket X_b \rrbracket_u, \llbracket Y_b \rrbracket_u)$ is permuted to $([U_i]_u, \llbracket V_i \rrbracket_u)$, then $U_i = 0$ since the element 1 in b-th column has been set to 0.

(b) Based on some RANK protocol, the server sorts $\frac{U_i}{V_i}$ $(1 \le i \le |\mathbf{B})|$ in the encrypted form. One straightforward way of constructing the RANK protocol is to combine an encrypted integer comparison protocol COM and any standard sorting algorithm. The COM protocol has slightly different semantics from that in the previous protocol in Section 4.4.1: user u has the private key and the service provider has two encrypted integers, at the end of the protocol the service provider learns the result.

(c) After the ranking, the server sends the ”Top-n” indexes (e.g. the permuted Top-n indexes) to user u, who can then recover the real Top-n indexes based on the permutation he has done.

4.5 Decentralized Friendship-based Protocol

In reality, *semi-honest* service provider is often viewed as a security weakness in protocol design. This motivates us to investigate privacy-preserving protocols in fully decentralized setting. Next, we first describe the setting and then present a decentralized single prediction protocol. Since we can extend the protocol to a Top-n variant in the same way as we have done in the centralized setting, we skip the details here.

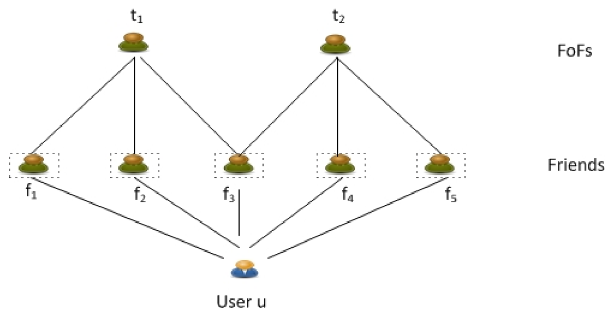


Figure 4.5.1: Decentralized System Structure

For simplicity, we assume that users are uniquely identified in the recommender system, and they share their social graph with their friends. In the initialization phase, user u generates his public/private key pair $(\mathsf{pk}_u, \mathsf{sk}_u)$ for a SWHE scheme. In addition, user u maintains a rating vector $\mathbf{R}_u$, his social graph, and assigns a weight $w_{u,f}$ to each of his friend $f \in \mathbf{F}_u$. All other users perform the same operations in this phase. Before going ahead, we want to point out that we choose a FoF as stranger in the following solution for the simplicity of description. In the view of user u, the topology is shown in Figure. 4.5.1. Due to the small world phenomenon, the population of FoFs can already be very large.

4.5.1 Decentralized Single Prediction Protocol

Next, we describe a protocol for user u to check whether $p_{u,i} \geq \tau$ according to Formula (4.3) in Section 4.3. It can be regarded as a decentralized version of the single prediction protocol from Section 4.4.1.

1. Based on the social graph (particularly his friend set F_u), user u chooses a stranger set T_u, consisting of his FoFs. He also chooses $t^* \in \mathsf{T}_u$. We further require that the every $f \in \mathsf{F}_u$ should have at least one friend in T_u.

2. User u generates a binary vector $\mathbf{I}_b$, which only has 1 for the b-th element, and broadcasts $[\![\mathbf{I}_b]\!]_u = \mathsf{Enc}(\mathbf{I}_b, \mathsf{pk}_u) = ([\![\mathbf{I}_b^{(1)}]\!]_u, \cdots, [\![\mathbf{I}_b^{(M)}]\!]_u)$ to his friends. He also sends $\mathsf{Enc}(w_{u,f}, \mathsf{pk}_u)$ to every user $f \in \mathsf{F}_u$.

3. With pk_u, $[\![\mathbf{I}_b]\!]_u$, $[\![w_{u,f}]\!]_u$ and $(\mathbf{R}_f, \mathbf{Q}_f)$, user f can compute the $[\![q_{f,b}]\!]_u$ and $[\![q_{f,b} \cdot (\mathbf{R}_f \cdot \mathbf{I}_b - \overline{r_f}) \cdot w_{u,f}]\!]_u$ in exactly the same way as in Section 4.4.1. User f then sends $[\![q_{f,b}]\!]_u$ and $[\![q_{f,b} \cdot (\mathbf{R}_f \cdot \mathbf{I}_b - \overline{r_f}) \cdot w_{u,f}]\!]_u$ to one of his friends in T_u. He also forwards $[\![\mathbf{I}_b]\!]_u$ and pk_u to the chosen friend.

4. For any $t \in \mathsf{T}_u$, he should receive $[\![\mathbf{I}_b]\!]_u$ and pk_u from at least one of his friend in F_u. If not, he can ask for such information from his friend. Then, he does the following.

   (a) Validate pk_u.

   (b) With pk_u and $(\mathbf{R}_t, \mathbf{Q}_t)$, every user t from $\mathbf{T}_u$ can compute $[\![q_{t,b}]\!]_u$ and $[\![q_{t,b} \cdot (\mathbf{R}_t \cdot \mathbf{I}_b - \overline{r_t})]\!]_u$ in exactly the same way as in Section 4.4.1.

   (c) Suppose that user t has received $[\![q_{f,b}]\!]_u$ and $[\![q_{f,b} \cdot (\mathbf{R}_f \cdot \mathbf{I}_b - \overline{r_f}) \cdot w_{u,f}]\!]_u$ for $f \in \mathsf{F}_u^-$ where $\mathsf{F}_u^- \subseteq \mathsf{F}_u$. He computes $\sum_{f \in \mathbf{F}_u^-} [\![q_{f,b}]\!]_u$ and $\sum_{f \in \mathbf{F}_u^-} [\![q_{f,b} \cdot (\mathbf{R}_f \cdot \mathbf{I}_b - \overline{r_f}) \cdot w_{u,f}]\!]_u$.

   (d) User t sends $[\![q_{t,b}]\!]_u$, $[\![q_{t,b} \cdot (\mathbf{R}_t \cdot \mathbf{I}_b - \overline{r_t})]\!]_u$, $\sum_{f \in \mathbf{F}_u^-} [\![q_{f,b}]\!]_u$ and $\sum_{f \in \mathbf{F}_u^-} [\![q_{f,b} \cdot (\mathbf{R}_f \cdot \mathbf{I}_b - \overline{r_f}) \cdot w_{u,f}]\!]_u$ to user t^*.

5. User t^* receives $[\![q_{t,b}]\!]_u$, $[\![q_{t,b} \cdot (\mathbf{R}_t \cdot \mathbf{I}_b - \overline{r_t})]\!]_u$, $\sum_{f \in \mathbf{F}_u^-} [\![q_{f,b}]\!]_u$ and $\sum_{f \in \mathbf{F}_u^-} [\![q_{f,b} \cdot (\mathbf{R}_f \cdot \mathbf{I}_b - \overline{r_f}) \cdot w_{u,f}]\!]_u$ from $t \in \mathbf{T}_u$. He then does the following.

(a) Compute $[\![n_T]\!]_u$, $[\![d_T]\!]_u$, $[\![n_F]\!]_u$, $[\![d_F]\!]_u$ and $[\![X]\!]_u$, $[\![Y]\!]_u$ in exactly the same way as in Section 4.4.1.

(b) Run a comparison protocol COM with user u for the latter to learn whether $\frac{X}{Y} \geq \tau - \overline{r_u}$.

4.5.2 Comparison to Centralized Protocol

In contrast to the centralized protocol from Section 4.4.1, the task of the semi-honest service provider is distributed to the "strangers", namely FoFs of user u. The overall computational complexity stays the same. The reason we have chosen the strangers to handle most of the computations is to reduce the complexity of the friends. In reality, the number of friends will be very limited, while the number of FoFs is much larger so that the chance a FoF is chosen is quite low.

If we assume trust can propagate through a chain of friends, then the strangers can be chosen more freely in the above solution. In comparison to the protocol from Section 4.4.1, this solution has the following advantages.

- The users do not need to semi-trust the service provider any more.
- User u can select the users (his friends and FoFs) to compute recommendations for himself. In order to do this, user u needs to maintain a social graph (at least his friends and FoFs).

However, it also has the following disadvantages.

- User u's FoFs need to perform more computations. Basically, the workload of the service provider has been shifted to them. This may become a heavy burden for the users.
- Users need to put more trust on their friends and FoFs, particularly on the user t^*. The users cannot leverage the service provider to blend their inputs anymore, and the trust has been shifted to user user t^*. In theory, this can be avoided by a secure multi-party computation protocol, but this will significantly increase the complexity.

Clearly, from the efficiency perspective, the centralized solution from Section 4.4.1 is more realistic in practice. In order to reduce the trust on the service provider, we can (at least) add two layers of validations on its behaviors. One is that, before participating in the protocol execution, a stranger can ask the service provider to provide a chain of friends so that he can validate the public key pk_u. The other is that user u can ask the service provider to prove that it has performed the required operations honestly.

4.6 Accuracy Properties of the Proposed Protocols

In this section, we investigate the recommendation accuracy of the prediction algorithms from Section 4.3, with respect to both centralized and decentralized settings where strangers are chosen differently therein. Because the 10-FMT dataset may be biased due to the fact that most of users don't post their movie ratings to Twitter, we also use MovieLens 100k dataset [45] with simulated friendships. Interestingly, the results align well in both datasets. In the experiments, we randomly split each data set into training set (80%) and testing set (20%). Note that in order to test all the users each time, instead of randomly splitting the original data sets in form of triplets (user_id, item_id, rating), we randomly split each user's rating history into training set (80%) and testing set (20%). In each test, a user's friends are randomly selected from his friend-set, the strangers are also randomly chosen. The MAE values summarized in the following tables are the mean value of their corresponding 5-fold cross validation.

4.6.1 Accuracy in Centralized Setting

With respect to the 10-FMT dataset, the MAE of the proposed recommendation algorithm, Section 4.3.1, is summarized in Table 4.6.1. Due to the fact that a user has limited number of friends in the 10-FMT dataset, we only compute MAE up to 50 friends. The column denotes the possible values of $\frac{\alpha}{\alpha+\beta}$ and the row denotes the possible values of $(|\mathbf{F}_u|, |\mathbf{T}_u|)$, where strangers are randomly sampled. Lower MAE implies more accurate recommendations.

	0.5	0.6	0.7	0.8	0.9	1.0
(10, 10)	0.6178	0.6197	0.6192	0.6299	0.6362	0.6388
(20, 10)	0.6140	0.6168	0.6156	0.6204	0.6208	0.6291
(30, 10)	0.6076	0.6090	0.6094	0.6169	0.6234	0.6371
(40, 10)	0.6073	0.6066	0.6104	0.6150	0.6215	0.6300
(50, 10)	0.6066	0.6053	0.6095	0.6138	0.6199	0.6289

Table 4.6.1: MAE on 10-FMT

With respect to the MovieLens 100k dataset, we define friends and strangers as follows. Given a user u, we first calculate the Cosine similarities with all other users and generate a neighborhood for user u. Then, we choose a certain number of users from the top-K_f most similar neighbors as the friends (In the experiments, $K_f = 250$.), and randomly choose a certain number of users from the rest as strangers. The MAE of the revised TW algorithm from Section 4.3.1 is summarized in Table 4.6.2. According to the accuracy results by Lemire and Maclachlan (in Table 1 of [67] where the values are MAE divided by 4), their smallest MAE is $0.752 = 0.188 \times 4$. We can get similar or lower MAE when $|\mathbf{F}_u| \geq 70$ by adjusting $\frac{\alpha}{\alpha+\beta}$.

	0.5	0.6	0.7	0.8	0.9	1.0
(10, 10)	0.8195	0.8112	0.8074	0.8104	0.8157	0.8290
(20, 10)	0.8115	0.8002	0.7964	0.7937	0.8028	0.8086
(30, 10)	0.8046	0.7932	0.7866	0.7822	0.7874	0.7952
(40, 10)	0.8000	0.7852	0.7779	0.7739	0.7770	0.7834
(50, 10)	0.7943	0.7800	0.7693	0.7666	0.7658	0.7728
(60, 10)	0.7913	0.7757	0.7640	0.7593	0.7601	0.7636
(70, 10)	0.7888	0.7715	0.7601	0.7536	0.7530	0.7572
(80, 10)	0.7856	0.7682	0.7561	0.7482	0.7470	0.7484
(90, 10)	0.7830	0.7665	0.7527	0.7445	0.7424	0.7428
(100, 10)	0.7815	0.7626	0.7492	0.7398	0.7371	0.7386

Table 4.6.2: MAE on MovieLens 100k

From the numbers in Table 4.6.1 and Table 4.6.2, there is a general trend that MAE decreases when friends number increases. We plot some columns of both tables for a better illustration, shown in Fig 4.6.1.

When the numbers of friends and strangers are fixed, the contribution factor $\frac{\alpha}{\alpha+\beta}$ also plays a role in determining recommendation accuracy. We plot some rows of both tables for a better illustration, shown in Fig 4.6.2. The MAE decreases when $\frac{\alpha}{\alpha+\beta}$ increases (i.e. friends has more contribution) on the MovieLens 100k dataset, while the MAE slightly increases when $\frac{\alpha}{\alpha+\beta}$ grows higher than 0.6 on the 10-FMT dataset.

4.6.2 Accuracy in Decentralized Setting

For the decentralized setting, we compute the MAE on both datasets and present them in Table 4.6.3 and Table 4.6.4 respectively. The MAE values are very close to those in Table 4.6.1 and Table 4.6.2, so that we can conclude that the recommendation accuracy is similar in both settings. It implies that sampling strangers from

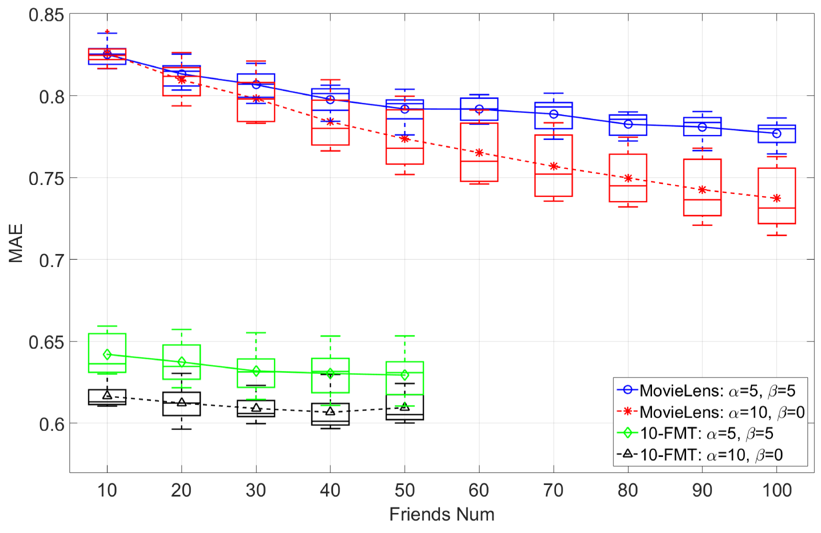


Figure 4.6.1: MAE evaluation with different friends number.

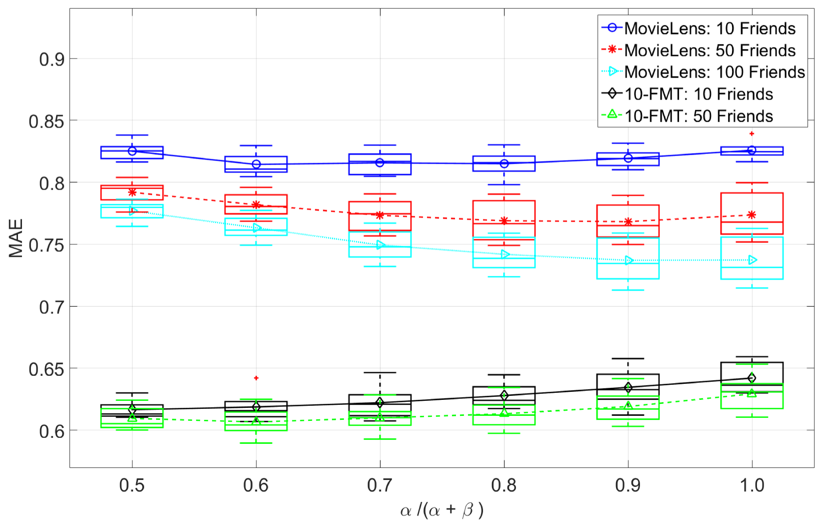


Figure 4.6.2: MAE evaluation with different $\frac{\alpha}{\alpha+\beta}$.

FoFs does not bring much accuracy gain with respect to both datasets.

	0.5	0.6	0.7	0.8	0.9	1.0
(10, 10)	0.6209	0.6168	0.6234	0.6289	0.6309	0.6465
(20, 10)	0.6135	0.6147	0.6164	0.6179	0.6275	0.6367
(30, 10)	0.6133	0.6085	0.6132	0.6188	0.6248	0.6309
(40, 10)	0.6124	0.6116	0.6110	0.6167	0.6233	0.6297
(50, 10)	0.6104	0.6084	0.6104	0.6147	0.6214	0.6301

Table 4.6.3: MAE on 10-FMT (Decentralized)

	0.5	0.6	0.7	0.8	0.9	1.0
(10, 10)	0.8181	0.8138	0.8132	0.8158	0.8188	0.8265
(20, 10)	0.8082	0.8034	0.7978	0.7994	0.8012	0.8123
(30, 10)	0.8026	0.7922	0.7879	0.7855	0.7885	0.7961
(40, 10)	0.7953	0.7862	0.7778	0.7763	0.7786	0.7826
(50, 10)	0.7917	0.7801	0.7726	0.7686	0.7688	0.7731
(60, 10)	0.7862	0.7747	0.7638	0.7625	0.7620	0.7664
(70, 10)	0.7854	0.7698	0.7604	0.7565	0.7532	0.7565
(80, 10)	0.7799	0.7663	0.7578	0.7502	0.7488	0.7489
(90, 10)	0.7781	0.7647	0.7524	0.7447	0.7407	0.7430
(100, 10)	0.7758	0.7603	0.7497	0.7406	0.7379	0.7377

Table 4.6.4: MAE on MovieLens 100k (Decentralized)

4.6.3 Comparison

Jeckmans et al. [57] proposed a similar solution, refer to as JPH protocol, in which only friends' data is counted in the recommendation computations. The prediciton process of JPH protocol is defined as

$$\begin{aligned} p_{u,b} &= \frac{\sum_{f \in \mathbf{F}_u} q_{f,b} \cdot r_{f,b} \cdot (\frac{w_{u,f}+w_{f,u}}{2})}{\sum_{f \in \mathbf{F}_u} q_{f,b} \cdot (\frac{w_{u,f}+w_{f,u}}{2})} \\ &= \frac{\sum_{f \in \mathbf{F}_u} r_{f,b} \cdot (w_{u,f} + w_{f,u})}{\sum_{f \in \mathbf{F}_u} q_{f,b} \cdot (w_{u,f} + w_{f,u})} \end{aligned} \tag{4.4}$$

where b denotes the item under prediction. the authors only discussed the security properties of their solutions without touching upon the performances. In this sections, we evaluate its accuracy performance on the two realword datasets.

Since strangers are not considered in the JPH prediction algorithm [57], we compute the MAEs by only considering friends. For comparison, we assume all the friends are rational, and let $\frac{w_{u,f}+w_{f,u}}{2}$ equal to the Cosine similarity between user u and friend f. With respect to the 10-FMT and MovieLens 100k dataset, the MAE results are summarized in Table 4.6.5 and 4.6.6 respectively. Clearly, their accuracy is much worse than our protocols which is mainly due to two reasons.

- JPH employs a very naive neighborhood-based method which can not capture users' rating preference. For example, some users prefer to give high ratings while some others lean to give low ratings.

- In reality, the data sets are very sparse and imbalanced. It may arise more serious cold-start problem to collaborative filtering techniques, including neighborhood-based method, if only using friends' rating information for predication.

Friends Num	10	20	30	40	50
MAE	2.5961	2.2464	2.0690	1.9677	1.9072

Table 4.6.5: MAE of JPH on 10-FMT

Friends Num	20	40	60	80	100
MAE	1.9021	1.49451	1.2978	1.1825	1.1018

Table 4.6.6: MAE of JPH on MoiveLens 100k

4.7 Security Analysis of the Proposed Protocols

In the threat model (Section 4.3.2), the service provider is assumed to be semi-honest, which means it will follow the protocol specification and does not participate in the protocol as a user. Moreover, a user trusts his friends to be semi-honest. As to communication channel among users, it is assumed that all communications are protected with respect to integrity and confidentiality (with forward secrecy). In the worst-case security model, it is assumed that some friends can be compromised.

The protocols from Section 4.4 and 4.5 are secure in both models based on the facts that all computations are done in the encrypted form under user u's public key and the comparison protocol is secure. It is worth noting that in these protocols the server does not need to generate any key pair for the SWHE scheme. As a result, the protocols are immune to key recovery attacks.

Next, we experimentally study the information leakages from recommendation outputs. We take the centralized protocols (where strangers are involved in the computation) as an example, and leave out the decentralized protocol which has similar results.

4.7.1 Inference from Outputs

Intuitively, the potential information leakages from recommendations depends on the global parameters α, β and the sizes of $\mathbf{F}_u$ and $\mathbf{T}_u$. If $\frac{\alpha}{\alpha+\beta}$ gets larger or the size of $\mathbf{T}_u$ gets smaller, then the inputs from friends contribute more to the final output of user u. This will in turn make inference attacks easier against the friends but harder against the strangers. In the protocol design, we explicitly prevent user u

from communicating with the strangers, therefore, user u will not trivially know whether a specific user t has been involved in the computation. The strangers are independently chosen in different protocol executions and the same stranger is unlikely to be involved in more than one executions, so that it is difficult for an attacker to leverage the accumulated information. Furthermore, we note the fact that there are many users in recommender systems but only 6 possible rating values for any item. This means that many users would give the same rating value $r_{t,b}$ for the item b. With respect to the single prediction protocol, even if $r_{t,b}$ is leaked, user u will not be able to link it to user t.

In our proposed algorithm, a friend f's contribution to $p_{u,b}$ is protected by the inputs from users in $\mathbf{F}_u \backslash f$ and the strangers in $\mathbf{T}_u$. Similarly, a stranger t's contribution to $p_{u,b}$ is protected by the inputs from users in $\mathbf{F}_u$ and strangers in $\mathbf{T}_u \backslash t$. We perform some experiments to show how a single friend or stranger influences the predicted rating values. We use the both the 10-FMT and MovieLens 100k datasets, and set $\frac{\alpha}{\alpha+\beta} = 0.8$. For illustration purpose, we only consider two settings, namely $(|\mathbf{F}_u|, |\mathbf{T}_u|) = (10, 10)$ and $(|\mathbf{F}_u|, |\mathbf{T}_u|) = (30, 10)$.

Take the setting $(|\mathbf{F}_u|, |\mathbf{T}_u|) = (10, 10)$ as an example, we perform the following experiment to test a friend's influence. In the experiment, we run 5-fold cross validation 50 times. In each 5-fold cross validation, we fix the friends of all users in the dataset by randomly selecting 11 friends for each user at the beginning, say each user has a fixed friend list L. Then for each user in the test set, the following procedure is carried out.

1. Randomly choose 10 strangers.

2. Randomly exclude 1 friend f_0 from the list L. Compute the predicted ratings of user u in the test set. Let the prediction vector be denoted as $\mathbf{P}_0$.

3. Randomly exclude 1 friend f_1 ($f_0 \neq f_1$) from the list L. Compute the predicted ratings of user u in the test set. Let the prediction vector be denoted as $\mathbf{P}_1$.

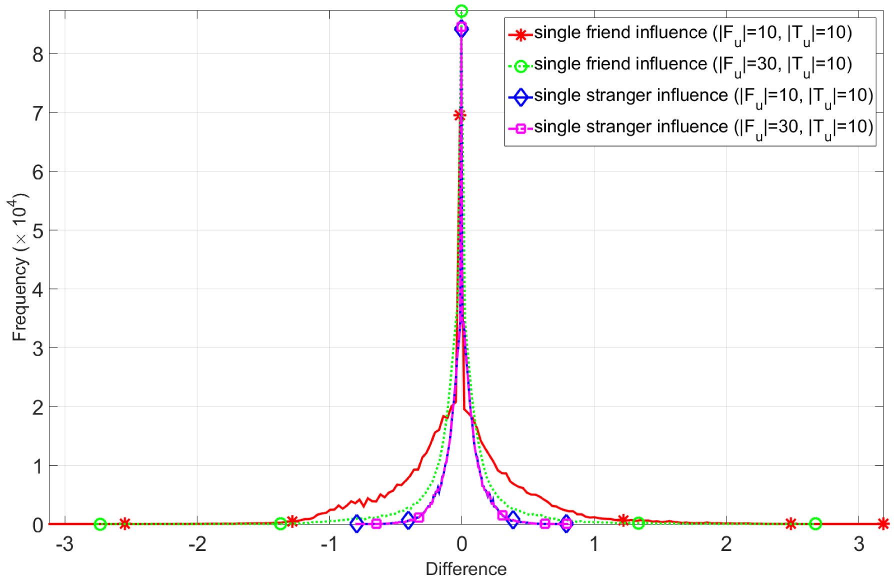


Figure 4.7.1: Single Friend (Stranger) Influence on 10-FMT Dataset

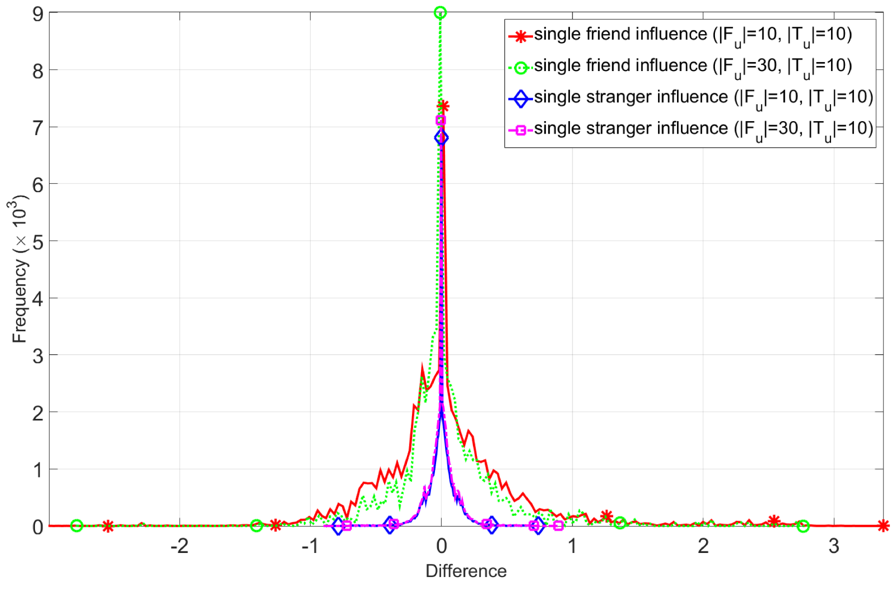


Figure 4.7.2: Single Firend (Stranger) Influence on MovieLens 100k Dataset

4. Compute the prediction difference vector as $\mathbf{P}_0 - \mathbf{P}_1$.

Experiments for testing a stranger's influence and for the $(|\mathbf{F}_u|, |\mathbf{T}_u|) = (30, 10)$ setting can be designed in a similar manner. After obtaining all prediction difference vectors $\mathbf{P}_0 - \mathbf{P}_1$ in the experiments, we plot the frequency of all difference values in Figure. 4.7.1 for the 10-FMT dataset and in Figure. 4.7.2 for the MovieLens 100k dataset. From the figures, it is obvious that an individual's influence to the output is quite small. In particular, a friend's influence becomes smaller when the friend set becomes larger. Another observation is that a stranger's influence is much smaller than a friend, but it stays almost the same when the friend set becomes larger.

Remark on Figure. 4.7.1. Statistically, $\geq$87% and $\geq$91% differences fall into the range $[-10^{-5}, 10^{-5}]$ w.r.t single friend influence testing and single stranger influence testing respectively. For clearly presenting those differences that fall out of the range $[-10^{-5}, 10^{-5}]$ and keeping the histogram structure, $\approx$98% of the differences that fall into the range $[-10^{-5}, 10^{-5}]$ have been removed.

Remark on Figure. 4.7.2. Statistically, $\geq$89% and $\geq$81% differences fall into the range $[-10^{-5}, 10^{-5}]$ w.r.t single friend influence testing and single stranger influence testing respectively. For clearly presenting those differences that fall out of the range $[-10^{-5}, 10^{-5}]$ and keeping the histogram structure, we remove $\approx$98% of the differences that fall into the range $[-10^{-5}, 10^{-5}]$.

4.8 Computational Complexity Analysis

In this section, we investigate the computational complexities of the protocols from Section 4.4.1 and 4.4.2. Since it is easy to infer the complexity of the decentralized protocol from the centralized one, we skip the details.

4.8.1 Asymptotic Analysis

With respect to the computational complexity of the proposed protocols, we first count the number of different computations required. For the single prediction protocol, the numbers of SWHE-related operations are listed in Table 4.8.1. COM will be executed once.

	Enc	Eval$(+,,)$	Eval$(\cdot,,)$
Friend	$2M+1$	$2M-1$	$2M+2$
Stranger	$2M+1$	$2M-1$	$2M+1$
Server	4	$2\lvert\mathbf{T}_u\rvert + 2\lvert\mathbf{F}_u\rvert - 3$	$\lvert\mathbf{F}_u\rvert + 6$
User u	$M + \lvert\mathbf{F}_u\rvert$	0	0

Table 4.8.1: Complexity of Single Prediction Protocol

For the Top-n protocol, the numbers of SWHE-related operations are listed in Table 4.8.2. In addition, if we instantiate the RANK protocol with the well-known Heapsort algorithm, the COM protocol needs to be executed $O(M \log M)$ times.

	Enc	Eval$(+,,)$	Eval$(\cdot,,)$
Friend	$2M$	0	M
Stranger	$2M$	0	0
Server	3	$2(\lvert\mathbf{T}_u\rvert+\lvert\mathbf{F}_u\rvert+M)M-5M$	$(2M+6+\lvert\mathbf{F}_u\rvert)M$
User u	$2M^2 + \lvert\mathbf{F}_u\rvert$	0	0

Table 4.8.2: Complexity of Top-n Protocol

4.8.2 Implementation Results

We instantiate a COM protocol based on that of Veugen [119] and evaluate its performance. In addition to the SWHE scheme based on YASHE [13], the protocol also relies Goldwasser-Micali scheme [44]. In both schemes, we set the bit-length of the prime number to be 512. We implement the Goldwasser-Micali scheme, which has the timing cost for Enc (1.5 μs), Dec (4.5 μs), based on an Intel(R) Core(TM) i7-5600U CPU 2.60GHz. In executing the COM protocol, the computation time for the client and the server is roughly 0.45 ms and 2.82 ms respectively.

We adopt the MovieLens 100k dataset where $M = 1682$ and set $(|\mathbf{F}_u|, |\mathbf{T}_u|) = (70, 10)$. We use the Microsoft SEAL library [28] based on YASHE scheme. The timing information of the SEAL lib is Enc (42 ms), Dec (41 ms), $\mathsf{Eval}(\cdot, ,)$ (305 ms), $\mathsf{Eval}(+, ,)$ (85 μs). The timing information of our protocols is shown in Table 4.8.3, and the source code is in [122].

	Friend	Stranger	Server	User u
Single	1.12	1.00	0.72	74.17
Top-n	141.22	140.55	1726446	236424

Table 4.8.3: Timing Numbers (Seconds)

Regardless the resource-constrained testing environment, it is clear that the Top-n protocol is very inefficient. The complexity mainly comes from the fact that we want to restrict user u to only learn the Top-n recommendations and prevent the server from learning any information. As such, there are two possible directions to relax the security guarantee and get better efficiency.

- One is to let user u learn more information (denoted as Relax-1 in Table 4.8.4). Referring to the protocol specification in Section 4.4.2, in stage 3, user u does not need to generate $\mathbf{M}_X, \mathbf{M}_Y$ and the server does not need to compute $([U_1]_u, [U_2]_u, \cdots, [U_M]_u)$ and $([\![V_1]\!]_u, [\![V_2]\!]_u, \cdots, [\![V_M]\!]_u)$. There is no need to perform the ranking, the server just sends $[\![X_b]\!]_u, [\![Y_b]\!]_u$ for every $1 \leq b \leq M$ to user u, who can decrypt these ciphertexts and obtain the Top-n recommendations.

- The other is to let the server learn how many items user u has rated (denoted as Relax-2 in Table 4.8.4). In addition, we need to assume that the strangers will not collude with the server. Referring to the protocol specification in Section 4.4.2, in stage 1 and 2, user u generates a random permutation for the items in the item set and share the permutation information with the friends and strangers. In stage 3, user u does not need to generate $\mathbf{M}_X, \mathbf{M}_Y$ and the server does not need to compute $([U_1]_u, [U_2]_u, \cdots, [U_M]_u)$ and

$(\llbracket V_1 \rrbracket_u, \llbracket V_2 \rrbracket_u, \cdots, \llbracket V_M \rrbracket_u)$. In stage 3, user u tells the server which items has been rated (the indices of these items have been permuted), and they interactively perform the ranking for the unrated items in the encrypted form as before.

	Friend	Stranger	Server	User u
Relax-1	141.22	140.55	1562	141.58
Relax-2	141.22	140.55	1610	10.46

Table 4.8.4: Timing Numbers (Seconds)

4.9 Related Work

Existing privacy-protection solutions can be generally divided into two categories. The cryptographic solutions (e.g. [5, 18, 86, 118]) often aim at securing the procedure of underlying recommender protocols, namely they do not consider the information leakage in the outputs. In this category, a typical method is to employ somewhat homomorphic encryption schemes and let all computations be done in encrypted form. Unfortunately, this will incur intolerable complexities and make the solutions impractical. Even though in the neighborhood-based systems, predictions are computed based on a small subset of users' data, a secure solution must compute the neighborhood privately in the first place, and this often introduces a lot of complexity. Moreover, many solutions (e.g. [86]) introduce additional semi-trusted servers which are difficult to be instantiated in reality. The data-obfuscation solutions (e.g. [109, 129]) rely on adding noise to the original data or computation results to protect users' inputs. These solutions usually do not incur complicated manipulations on the users' inputs, so that they are much more efficient. The drawback is that they often lack rigorous privacy guarantees and downgrade the recommendation accuracy to some extent.

In order to improve the efficiency of privacy-preserving recommender systems, one typical approach is to design more efficient cryptographic tools. However, even if the speedup is significant in cryptographic sense, it often does not result in practical recommender systems. This is due to the large underlying user populations, which make model training and neighborhood selection unrealistic even with efficient cryptographic tools. Recently, Jeckmans et al. proposed an interesting solution direction in [57], where they proposed the concept of friendship-based recommender system and gave solutions based on somewhat homomorphic encryption schemes. The rationale behind their concept is the following.

- In order to avoid the computationally-cumbersome neighborhood selection step in neighborhood-based recommender systems, the solution leverages auxiliary social network information of users. This approach reduces the amount of data used in computing predictions significantly.
- Trust is a very subtle issue. Friends may trust each other in the sense that their peers will not collude with a third party to leak their information. If a collusion is discovered, then their relationship can be broken. Moreover, some information may be sensitive among friends, but not with strangers. For instance, if a user has watched a porn movie, then disclosing this to his friends may make him embarrassed, but disclosing it to a stranger may not cause any harm. This motivates the adoption of homomorphic encryption to secure the computation.

4.10 Conclusion and Future Work

In the chapter, we presented a friendship-based recommender system which allows computing recommendations with the data from a few of users. We first demonstrated that our proposed recommender system allows a promising accuracy performance, then we implemented a secure protocol based on this system, with resort to a somewhat (fully) homomoprhic encryption scheme. This recommender system requires a much fewer number of users (for recommendation computation) than existing recommendation algorithms do, significantly improving the efficiency performance. We have also provided detailed analysis to recommendation accuracy, inference attacks, and computational complexities. The idea of introducing randomly selected strangers to prevent information leakages from the output share some similarity with the differential privacy based approach [76] and the differential identifiability approach [66]. A more rigorous comparison remains as an interesting future work, particularly in the line of the works from [11, 29]. With respect to accuracy analysis, it is an interesting future work to perform a study on an unbiased real-world dataset.

5 CryptoRec: Privacy-preserving Recommendation as a Service

5.1 Introduction

Machine Learning as a Service (MLaaS) has become increasingly popular, with the progress of cloud computing. As the recommender system is always one of the most important machine learning tasks, deploying cloud based recommendation service is definitely an important topic. In fact, commercial applications of Recommendation as a Service have existed, such as Amazon [6] and Recombee [101]. On the one hand, a user can efficiently get preferred products from a vast number of items due to the recommender system; on the other hand, the user data is exposed to the service provider and can be abused [31, 82]. As such, it results in immediate privacy risks to the user. In this chapter, we study how to obtain efficient and accurate recommendation services while preserving data privacy. As an illustrative example, consider the following scenario:

A user (client) with some private data (e.g., ratings) would like to buy a recommendation service to efficiently figure out the most favorable products from a large number of potential candidates. A service provider (e.g., Amazon) has already collected a large database of ratings given by its users on sold items and wishes to monetize its data by selling Recommendation as a Service (RaaS). Different from existing recommender systems, in our scenario the client is unwilling to expose her data to the service provider due to the worries of privacy leakage. At the same time, commercial concerns or requirements may prevent the service provider from releasing its trained recommendation model to the public. In addition, releasing a trained model may also bring privacy risks to the users in the service provider's database.

We can formalize the above scenario as a secure two-party computation (2PC) protocol. We first describe the Recommendation as a Service (RaaS) as a 2PC protocol, where on one side we have the Server (service provider) with its training data and on the other side the Client with her input. When the protocol terminates, the Client learns a prediction for her input. For obvious reasons, the protocol is only as useful to the Client as the accuracy of the predictions. Then we define the security of the 2PC protocol from two aspects: (1) the Client should only learn the predictions (including what can be inferred from the predictions); (2) the Server should not learn anything about the Client's input. General cryptographic primitives such as secure multi-party computation (SMC) and homomorphic encryption (HE) are immediate candidates to overcome these security concerns.

State-of-the-art recommender systems often rely on non-linear operations, or require training the recommendation model with the Client's data [64, 78, 132, 133]. Generic solutions usually come at a prohibitive cost in terms of efficiency. While improving cryptographic tools (e.g., HE or SMC) is one typical way to achieve more efficient privacy-preserving solutions, unfortunately, the improvement is usually far from satisfactory to make these solutions practical enough. Recently, CryptoNets [43] and MiniONN [72] have been proposed for privacy preserving neural networks based MLaaS, the scenario of which is similar to ours. Moreover, they assume machine learning models have been pre-trained (the Client's data is not required to be existing in the training set). The primary contribution of CryptoNets and MiniONN is how to efficiently compute non-linear operations on encrypted data. Instead of using state-of-the-art non-linear activation functions such as ReLu ($relu(x) = max(0, x)$), CryptoNets proposed using a square activation function ($f(x) = x^2$) to avoid non-linear operations, to facilitate evaluating neural networks on encrypted data. This approach may result in a significant accuracy loss [72]. MiniONN introduced a multi-round interactive protocol based on HE and garbled circuits [62], in which non-linear operations were computed by interactions between the Server and the Client. This method requires the two parties to be online constantly, which may increase the difficulty of using MLaaS.

Our contributions. We tackle this problem from the direction of designing crypto-friendly machine learning algorithms, so that we can achieve efficient solutions by directly using existing cryptographic tools. In particular, we propose CryptoRec, a new non-interactive secure 2PC protocol for RaaS, the key technical innovation of which is an HE-friendly recommender system. This recommendation model possesses two important properties: (1) It uses only addition and multiplication operations, so that it is straightforwardly compatible with HE schemes. With this property, CryptoRec is able to complete recommendation computations without requiring the Server and the Client to be online continuously. Simply put, the Client sends her encrypted rating vector to the Server, then the Server computes recommendations with the Client's input and returns the results in an encrypted form. In addition to this, there is no other interaction between the two parties; (2) It can automatically extract personalized user representations by aggregating pre-learned item features, that we say the model has an item-only latent feature space. This property allows the Server with a pre-trained model to provide recommendation services without a tedious re-training process, which significantly improves the efficiency performance. Note that the Client's data is not in the Server's database which is used for model training.

CryptoRec is able to produce recommendations in a direct mode (using only a pre-trained model learned on the Server's database which does not contain the Client's data) or in a re-training mode (where the model is first re-trained with the Client's input before computing recommendations). The re-training mode produces slightly more accurate predictions. In the direct mode, we can instantiate our protocol with an additive HE scheme such as the very efficient Paillier cryptosystem [91]. We test both modes of CrytoRec on MovieLens-1M (ml1m)[45], Netflix (netlfix) [84] and Yahoo-R4 (yahoo) [1] public datasets. Experiment results show that the direct mode allows the Server with thousands of items to privately answer a prediction query in a few seconds on a single PC. To re-train the model with the Client's input, we need a limited number of homomorphic additive and multiplicative operations. Therefore, we must rely on a Somewhat HE scheme (SWHE) [37]. Besides the advantage that our solution relies only on linear operations and converges in a very few numbers of iterations, the accuracy of the predictions produced by our model is less than 2% away from those achieved by the most accurate collaborative learning algorithms known to date (depending on the datasets). In practice, the Client can choose either of the two modes, according to her preference on the trade-off between accuracy and efficiency.

As a byproduct, the CryptoRec model naturally achieves transferability, in the sense that we can easily transfer the knowledge learned from one dataset to another if they share the same item set. We experimentally demonstrate the transferability of our model and discuss how it facilitates recommendations as a service in terms of both utility (i.e. accuracy and efficiency) and privacy.

5.2 CryptoRec

In this section, we present CryptoRec, a non-interactive secure 2PC protocol built on top of a new homomorphic encryption-friendly recommender system, referred to as CryptoRec's model. In Section 5.2.1, we introduce CryptoRec's model. In Section 5.2.2 we explain how to train the model and learn the parameters Θ. Finally, in Section 5.2.3, we combine the prediction procedure of our model with homomorphic encryption. This gives rise to our CryptoRec protocol. We also consider a second variant of the protocol in which the model parameters Θ are re-trained before computing recommendations. The re-training occurs in encrypted form, therefore it results in better accuracy without compromising security. Naturally, the computational cost on the Server side is considerably heavier.

5.2.1 CryptoRec's Model

Existing collaborative filtering (CF) technologies require non-linear operations or re-training with the Client's data [64, 78, 132, 133]. Directly applying the existing CFs to encrypted data leads to severe efficiency problems. To address this issue, we propose CryptoRec's model, a new homomorphic encryption friendly recommender system. It models user-item interaction behaviors in an item-only latent feature space. This means that the user features do not exist in the latent feature space, the model will automatically compute the user features by aggregating pre-learned item features. This property allows the Server with a pre-trained model to provide recommendations for the Client without having to re-train the model with the Client's data. Algebraic operations in CryptoRec's model are constrained to only additions and multiplications, thus CryptoRec's model is straightforwardly compatible with homomorphic encryption schemes.

We exploit the fact that a user profile is essentially identified by items that the user has rated, to construct personalized user features in an item-only latent feature space. In particular, we model the personalized user features $\mathbf{p}_u$ by aggregating pre-learned item features $\mathbf{Q} = \{\mathbf{q}_i\}_{i=1}^{m}$ as follows,

$$\mathbf{p}_u = \mathbf{r}_u\mathbf{Q} \tag{5.1}$$

therefore we can approximate an observed rating r_{ui} by

$$r_{ui} \approx \hat{r}_{ui} = \underbrace{(\mathbf{r}_u\mathbf{Q})}_{\mathbf{p}_u}\mathbf{q}_i^T \tag{5.2}$$

Using only a single latent feature space **Q** to model a large number of ratings often leads to an information bottleneck. To address this issue, we relax the item features which were used to construct user features **P**, and redefine the Equation (5.2) as

$$r_{ui} \approx \underbrace{(\mathbf{r}_u \mathbf{A})}_{\mathbf{p}_u} \mathbf{q}_i^T \tag{5.3}$$

Note that $\mathbf{A} \in \mathbb{R}^{m \times d}$ is a new item feature space.

We now have the basic form of CryptoRec's model which has an item-only latent feature space and relies only on addition and multiplication operations. However, it is not robust enough in practice due to the high variance of individual user or item behaviors, commonly known as biases. For example, real-world datasets exhibit large systematic tendencies for some users to give higher ratings than others, and for some items to receive higher ratings than others [64]. To address this issue, a common approach is to identify the portion of these ratings that individual user or item biases can explain, subjecting only the true interaction portion of the ratings to latent factor modeling [64]. The user and item biases are often approximated by

$$b_{ui} = \mu + b_u + b_i \tag{5.4}$$

where $\mu = \frac{\sum_{(u,i)\in \mathbf{R}} r_{ui}}{N}$ is the global rating average, N is the number of observed ratings. b_u and b_i approximate user bias and item biases, respectively. To obtain b_u and b_i, we can either compute $b_u = \bar{r}_u - \mu$ and $b_i = \bar{r}_i - \mu$ [64], or directly learn their values from a dataset [63]. The former ignores the global effects upon a single user or item; the latter models both the individual behaviors and global effects, but sometimes it leads to an early overfitting. To maintain both reliability and accuracy, we separately model the individual behaviors and global effects as follows

$$b_{ui} = \mu + b_u + b_i + b_u^* + b_i^* \tag{5.5}$$

where b_u and b_i are computed as $b_u = \bar{r}_u - \mu$ and $b_i = \bar{r}_i - \mu$. b_u^* and b_i^* are the parameters directly learned from the dataset to capture only the impact of global effects upon a single user and item, respectively.

We combine the biases approximator (Equation (5.5)) and the user-item interaction approximator (Equation (5.3)) to formalize the final CryptoRec's model as following,

$$r_{ui} \approx \hat{r}_{ui} = \underbrace{\mu + b_u + b_i + b_u^* + b_i^*}_{biases} + \underbrace{(\mathbf{r}_u \mathbf{A})\mathbf{q}_i^T}_{interaction} \tag{5.6}$$

As such, the user preference estimation is separated into two parts: biases approximator and user-item interaction approximator. This allows only the true user-item interaction being modeled by the factor machine (i.e., Equation (5.3)). The model parameters of CryptoRec's model are $\Theta = \{\mathbf{A}, \mathbf{Q}, \mathbf{b}_u^*, \mathbf{b}_i^*\}$ [1], where $\mathbf{b}_u^* = \{b_u^*\}_{u=1}^n$, $\mathbf{b}_i^* = \{b_i^*\}_{i=1}^m$.

5.2.2 Training

The model parameters $\Theta = \{\mathbf{A}, \mathbf{Q}, \mathbf{b}_u^*, \mathbf{b}_i^*\}$ are learned by solving the regularized least squares objective function,

$$\begin{aligned} \mathcal{L} = & \sum_{u=1}^{n} ||(\hat{\mathbf{r}}_u - \mathbf{r}_u) \cdot \varphi_u||^2 \\ & + \lambda \cdot (||\mathbf{A}||^2 + ||\mathbf{Q}||^2 + ||\mathbf{b}_u^*||^2 + ||\mathbf{b}_i^*||^2) \end{aligned} \tag{5.7}$$

where $\hat{r}_{ui}$ is defined in Equation (5.6), $\varphi_u = \{\varphi_{ui}\}_{i=1}^m$ and $(\hat{\mathbf{r}}_u - \mathbf{r}_u) \cdot \varphi_u$ denotes $\{(\hat{r}_{ui} - r_{ui})\varphi_{ui}\}_{i=1}^m$. If user u rated item i, then $\varphi_{ui} = 1$, otherwise, we let $\varphi_{ui} = 0$ and $r_{ui} = 0$. We use φ_{ui} to remove the (incorrect) gradients computed on unobserved ratings. The constant λ controls the extent of regularization. When performing training on plaintext dataset, the Server can compute φ_{ui} by itself, avoiding the unnecessary gradient computations on unobserved ratings.

As shown in Equation (5.6), CryptoRec's model is in fact a two-layer network. The first layer outputs the user feature vector $\mathbf{p}_u = \mathbf{r}_u\mathbf{A}$ and the second layer integrates the user features, item features and biases to estimate the user prefer-

[1] For the convenience of notation in describing the algorithms later on, we omit $\{\mu, \mathbf{b}_u, \mathbf{b}_i\}$ from the model parameters Θ in favor of a slightly more succinct notation. Note that $\{\mu, \mathbf{b}_u, \mathbf{b}_i\}$ are not learned by training procedure either.

ences. Back-propagation [65] is a standard method used in neural networks to calculate a gradient that is needed in the calculation of the values of parameters in the network. Simply put, we first compute the predictions given the input data (i.e., a forward pass); and then we calculate the total error according to the objective function (e.g., Equation (13)). Lastly, we compute the gradient of each trainable parameter using the error back-propagated through the network layers (i.e., a backward pass). Using the back-propagation method, we have the gradient of each model parameter of CrypotRec's model as follows,

$$
\begin{aligned}
\Delta\mathbf{A} &= \frac{\partial\mathcal{L}}{\partial\mathbf{p}_u}\cdot\frac{\partial\mathbf{p}_u}{\partial\mathbf{A}} = [(\mathbf{e}_u\cdot\varphi_u)\mathbf{Q}] \circledast \mathbf{r}_u^T + \lambda\cdot\mathbf{A}\\
\Delta\mathbf{q}_i &= \frac{\partial\mathcal{L}}{\partial\mathbf{q}_i} = \varphi_{ui}\cdot(e_{ui}\cdot(\mathbf{r}_u\mathbf{A}) + \lambda\cdot\mathbf{q}_i)\\
\Delta b_u^* &= \frac{\partial\mathcal{L}}{\partial b_u^*} = \mathbf{e}_u\varphi_u + \lambda\cdot b_u^*\\
\Delta b_i^* &= \frac{\partial\mathcal{L}}{\partial b_i^*} = \mathbf{e}_i\varphi_i + \lambda\cdot b_i^*
\end{aligned}
\tag{5.8}
$$

where $e_{ui} = \hat{r}_{ui} - r_{ui}$, $\mathbf{e}_u = \{e_{ui}\}_{i=1}^m$, $\mathbf{e}_i = \{e_{ui}\}_{u=1}^n$, and $\mathbf{e}_u\cdot\varphi_u = \{e_{ui}\cdot\varphi_{ui}\}_{i=1}^m$. $\circledast$ denotes outer product [2]. We randomly divide the dataset into multiple batches. In the training phase, we compute gradient by batch and update the model parameters by moving in the opposite direction of the gradient (i.e., gradient descent optimization algorithm). Algorithm 4 outlines the model training procedure. The learning rate η is used to control the speed of model updates. Note that the training procedure only relies on addition and multiplication operations.

5.2.3 Two Secure Protocols

In this section, we introduce two CryptoRec secure protocols. In the first protocol, the Server uses pre-trained model parameters Θ and directly takes as input the Client's encrypted rating vector to compute recommendations. In the second

[2] Given two vectors $\mathbf{x}^{1\times m}$ and $\mathbf{y}^{n\times 1}$, $(\mathbf{x}\circledast\mathbf{y})_{ij} = x_i y_j$

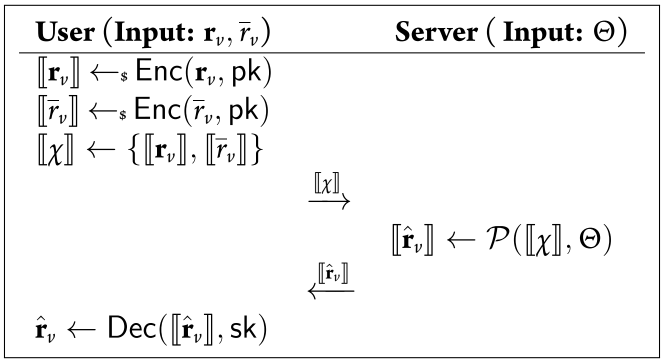
Algorithm 4 CryptoRec's model training procedure $\mathcal{T}$

Input: Rating $\mathbf{R}$, rating indicator Φ , user mean ratings $\bar{\mathbf{r}}_u = \{\bar{r}_u\}_{u=1}^n$,
$\Theta = \{\mathbf{A}^{(0)}, \mathbf{Q}^{(0)}, \mathbf{b}_u^{*(0)}, \mathbf{b}_i^{*(0)}\}$
Output: Optimized $\Theta = \{\mathbf{A}^{(K)}, \mathbf{Q}^{(K)}, \mathbf{b}_u^{*(K)}, \mathbf{b}_i^{*(K)}\}$

```
1: procedure $\mathcal{T}(\{\mathbf{R}, \Phi, \bar{\mathbf{r}}_u\}, \Theta)$
2:     for $k \leftarrow \{1, 2, \cdots, K\}$ do
3:         $\mathbf{A}^{(k)} \leftarrow \mathbf{A}^{(k-1)} - \eta \cdot \Delta\mathbf{A}^{(k-1)}$          ▷ $\eta$: learning rate
4:         $\mathbf{Q}^{(k)} \leftarrow \mathbf{Q}^{(k-1)} - \eta \cdot \Delta\mathbf{Q}^{(k-1)}$
5:         $\mathbf{b}_u^{*(k)} \leftarrow \mathbf{b}_u^{*(k-1)} - \eta \cdot \Delta\mathbf{b}_u^{*(k-1)}$
6:         $\mathbf{b}_i^{*(k)} \leftarrow \mathbf{b}_i^{*(k-1)} - \eta \cdot \Delta\mathbf{b}_i^{*(k-1)}$
7:     return $\Theta = \{\mathbf{A}^{(K)}, \mathbf{Q}^{(K)}, \mathbf{b}_u^{*(K)}, \mathbf{b}_i^{*(K)}\}$
```

protocol, the Server re-trains the model parameters Θ before computing recommendations. For the sake of clarity, we denote the Client as v in this section.

Secure protocol with a pre-trained model. Figure 5.2.1 describes the security protocol for prediction with pre-trained model parameters Θ. The Client v sends $[\![\mathbf{r}_v]\!]$ and $[\![\bar{r}_v]\!]$ to the Server, which executes the prediction process $\mathcal{P}$ (described in Figure 5.2.1) and returns the encrypted results $\hat{\mathbf{r}}_v$.



$$\begin{array}{lcl}
\textbf{User } (\textbf{Input: } \mathbf{r}_v, \bar{r}_v) & & \textbf{Server } (\textbf{ Input: } \Theta) \\
\hline
[\![\mathbf{r}_v]\!] \leftarrow_\$ \mathsf{Enc}(\mathbf{r}_v, \mathsf{pk}) & & \\
[\![\bar{r}_v]\!] \leftarrow_\$ \mathsf{Enc}(\bar{r}_v, \mathsf{pk}) & & \\
[\![\chi]\!] \leftarrow \{[\![\mathbf{r}_v]\!], [\![\bar{r}_v]\!]\} & & \\
& \xrightarrow{[\![\chi]\!]} & \\
& & [\![\hat{\mathbf{r}}_v]\!] \leftarrow \mathcal{P}([\![\chi]\!], \Theta) \\
& \xleftarrow{[\![\hat{\mathbf{r}}_v]\!]} & \\
\hat{\mathbf{r}}_v \leftarrow \mathsf{Dec}([\![\hat{\mathbf{r}}_v]\!], \mathsf{sk}) & &
\end{array}$$

Figure 5.2.1: CryptoRec with pre-trained $\Theta = \{\mathbf{A}, \mathbf{Q}, \mathbf{b}_u^*, \mathbf{b}_i^*\}$

We present the prediction process $\mathcal{P}$ of CryptoRec's model in Algorithm 5. The

Algorithm 5 CryptoRec's model prediction procedure $\mathcal{P}$

Input: Ratings $[\![\mathbf{r}_v]\!], [\![\bar{r}_v]\!]$, $\Theta = \{\mathbf{A}, \mathbf{Q}, \mathbf{b}_u^*, \mathbf{b}_i^*, \mu, \mathbf{b}_u, \mathbf{b}_i\}$
Output: Recommendations $[\![\hat{\mathbf{r}}_v]\!]$

```
1:  procedure $\mathcal{P}(\{[\![\mathbf{r}_v]\!], [\![\bar{r}_v]\!]\}, \Theta)$
2:      if $b_v^* \notin \mathbf{b}_u^*$ then                          ▷ $b_v^* \in \mathbf{b}_u^*$ if re-traininng the $\Theta$
3:          $b_v^* \leftarrow \frac{\sum_{u=1}^{n} b_u^*}{n}$
4:      $[\![\mathbf{p}_v]\!] \leftarrow [\![\mathbf{r}_v]\!]\mathbf{A}$                  ▷ HE dot-product using $\odot$ and $\oplus$
5:      for $i \leftarrow [1, 2, \cdots, m]$ do
6:          $[\![x_1]\!] \leftarrow (b_i + b_i^* + b_v^*) \oplus [\![\bar{r}_v]\!]$            ▷ $b_v \leftarrow \bar{r}_v - \mu$
7:          $[\![x_2]\!] \leftarrow [\![\mathbf{p}_v]\!]\mathbf{q}_i^T$
8:          $[\![\hat{r}_{vi}]\!] \leftarrow [\![x_1]\!] \oplus [\![x_2]\!]$
9:          $[\![\hat{\mathbf{r}}_v]\!][i] \leftarrow [\![\hat{r}_{vi}]\!]$
10:     return $[\![\hat{\mathbf{r}}_v]\!]$
```

computation is straightforward since CryptoRec's model contains only addition and multiplication operations, as we can observe in Equation (5.6). The inputs of this algorithm are the Client's encrypted rating vector $[\![\mathbf{r}_v]\!]$, the average rating $[\![\bar{r}_v]\!]$ and model parameters Θ. Since the Client's b_v^* is unknown to the Server, b_v^* is set to the average value of $\mathbf{b}_u^*$ (line 2-3, Algorithm 5).

Secure protocol with re-training. In order to achieve the most accurate predictions, we introduce a re-training process to the CryptoRec protocol, shown in Figure 5.2.2. Compared to the secure protocol without a re-training step (using only a pre-trained model, Figure 5.2.1), there are two differences: The first one is that, besides $[\![\mathbf{r}_v]\!]$ and $[\![\bar{r}_v]\!]$, the user also sends the encrypted indication vector $[\![\varphi_v]\!]$ to the Server, which will be used in the training procedure $\mathcal{T}$. The second one is that, before computing recommendations using $\mathcal{P}$ (Algorithm 5), the Server re-trains the model parameters Θ with the Client's inputs.

The training procedure $\mathcal{T}$ is described in Algorithm 4, takes advantage of homomophic properties of the encryption scheme. It is worth stressing that in the re-training protocol, we re-train the model parameters Θ with only the Client's

$$\begin{array}{lcl}
\textbf{User } (\textbf{Input: } \mathbf{r}_v, \varphi_v, \bar{r}_v) & & \textbf{Server } (\textbf{ Input: } \Theta) \\
\hline
[\![\mathbf{r}_v]\!] \leftarrow_{\$} \mathsf{Enc}(\mathbf{r}_v, \mathsf{pk}) & & \\
[\![\varphi_v]\!] \leftarrow_{\$} \mathsf{Enc}(\varphi_v, \mathsf{pk}) & & \\
[\![\bar{r}_v]\!] \leftarrow_{\$} \mathsf{Enc}(\bar{r}_v, \mathsf{pk}) & & \\
[\![\chi]\!] \leftarrow \{[\![\mathbf{r}_v]\!], [\![\varphi_v]\!], [\![\bar{r}_v]\!]\} & & \\
 & \xrightarrow{[\![\chi]\!]} & \\
 & & [\![\Theta]\!] \leftarrow \mathcal{T}([\![\chi]\!], \Theta) \\
 & & [\![\chi_1]\!] \leftarrow \{[\![\mathbf{r}_v]\!], [\![\bar{r}_v]\!]\} \\
 & & [\![\hat{\mathbf{r}}_v]\!] \leftarrow \mathcal{P}([\![\chi_1]\!], [\![\Theta]\!]) \\
 & \xleftarrow{[\![\hat{\mathbf{r}}_v]\!]} & \\
\hat{\mathbf{r}}_v \leftarrow_{\$} \mathsf{Dec}([\![\hat{\mathbf{r}}_v]\!], \mathsf{sk}) & &
\end{array}$$

Figure 5.2.2: CryptoRec with re-training $\Theta = \{\mathbf{A}, \mathbf{Q}, \mathbf{b}_u^*, \mathbf{b}_i^*\}$

data, not the Server's dataset. For efficiency reasons, the Server should pre-train the model parameters Θ on its dataset. Note that after the re-training process, the model parameters Θ are encrypted. So the related algebraic operations in the prediction process $\mathcal{P}$ should be also updated to their corresponding homomorphic operations.

The data security and the correctness of algebraic operations are guaranteed by HE primitives straightforwardly. The client can (always) generate a new pair of secret/public keys to encrypt its data when requiring recommendation services. The server trains a randomly initialized CryptoRec model with a stochastic method, so the learned model is non-deterministic.

5.3 Experiment Setup

We evaluate the accuracy and efficiency performances of CryptoRec on the rating prediction task and compare CryptoRec with several state-of-the-art collaborative filtering algorithms, including item-based NBM (I-NBM) [25], biased matrix factorization (BiasedMF) [64], user-based AutoRec (U-AutoRec) [106] and item-based AutoRec (I-AutoRec) [106]. We test these models on three datasets which are widely used for recommender systems performance evaluation, as shown in

Table 5.3.1. The dataset ml1m [45] contains 1 million ratings; yahoo [1] contains 0.21 million ratings; For netflix [84] dataset, we select 11,000 users who have given 1.2 million ratings to 4,768 items, where each user has at least 30 ratings. The testbed is a single PC with 8 Intel (R) Xeon(R) CPUs running at 3.5 GHz, with 32 GB of RAM, running the Ubuntu 16.04 operating system. All the 8 CPUs are used in the experiments.

	user #	item #	density	scale
netflix	11,000	4,768	2.17%	[1,5]
ml1m	6,040	3,952	4.2%	[1,5]
yahoo	7,637	3,791	0.72%	[1,5]

Table 5.3.1: Datasets used for benchmarking

5.3.1 Dataset Splitting

For each dataset, we randomly split all the users into a training set (80%) and a validation set (20%), and then we randomly divide each user data vector of the validation set into a feeding set (90%) and a testing set (10%). The training set simulates the Server's dataset, the feeding set simulates the rating data of the Client, and the testing set is used for accuracy evaluation. In the experiments, the Server trains recommendation models with its dataset. The Client sends its rating data vector to the Server, as a query, to get recommendations. For the models which have to be trained with the Client's data (the feeding set), we directly append the feeding set to the training set. These models, which require training from scratch with the Client's input, are identified in Section 5.3.2. For all the models, we repeat the accuracy evaluation experiments five times on each dataset. The root mean square error (RMSE) is adopted as the accuracy metric,

$$RMSE = \sqrt{\frac{\sum_{(u,i)\in\mathcal{D}}(\hat{r}_{ui} - r_{ui})^2}{|\mathcal{D}|}}$$

where $\mathcal{D}$ is the testing set, $|\mathcal{D}|$ is the number of ratings in the testing set. The lower the RMSE value, the higher the accuracy performance is.

5.3.2 Remarks on Model Training

	NBM	MF	Neural Network Based
w/o Client	I-NBM[25]	$\varnothing$	U-AutoRec[106],[90, 104, 113]
w/ Client	U-NBM[25]	BiasedMF[64],[63, 78]	I-AutoRec[106], [50, 60, 123, 133]

Table 5.3.2: Remarks on model training

By investigating recommender systems which aim to provide accurate rating predictions, we informally classify these models into two categories as shown in Table 5.3.2. the category "w/o Client" contains the models which allow offering recommendations with a pre-trained model while the Client's private data is not in the training set; the category "w/ Client" includes the models which have to be trained or re-trained with the Client's data. We refer interested readers to the two comprehensive reviews [114, 132] for more details.

The models which fall into "w/ Client" category often have one or both of the two following characteristics,

- User and item features are jointly learned in the training phase, such as MF and its variants [63, 64, 78].
- The input is an item rating vector ($\mathbf{r}_i$), such as U-NBM [25] and I-AutoRec [106].

The models in the category of "w/o Client" often take as input a user preference vector (e.g., $\mathbf{r}_u$). The personalized user features are automatically captured in the prediction phase, such as I-NBM [25], U-AutoRec [106], and our proposed CryptoRec's model.

We select I-NBM, BiasedMF (the representatives of traditional recommender systems), and U-AutoRec, I-AutoRec (the representatives of neural network based recommender systems) as the comparison baselines.

	netflix		ml1m		yahoo	
	RMSE	loss %	RMSE	loss %	RMSE	loss %
I-NBM	0.9115±0.007	9.4	0.8872±0.012	6.0	0.9899±0.017	0.2
U-AutoRec	0.9762±0.012	17.1	0.9526±0.007	13.9	1.0621±0.014	7.5
CryptoRec	**0.8586±0.005**	**3.0**	**0.8781±0.007**	**4.9**	**0.9888±0.011**	**0.1**
I-AutoRec	*0.8334±0.006*	*0*	*0.8367±0.004*	*0*	*0.9880±0.015*	*0*

Table 5.3.3: Accuracy comparison with pre-trained models. I-AutoRec is the accuracy benchmark.

5.3.3 Accuracy Benchmark

Without considering privacy, the model I-AutoRec achieves state-of-the-art accuracy (RMSE) performance [106]. As such, we adopt I-AutoRec as the accuracy benchmark model and train it from scratch in a standard machine learning setting. Table 5.3.4 presents the accuracy performance of I-AutoRec on the selected datasets.

	netflix	ml1m	yahoo
I-AutoRec	0.8334±0.006	0.8367±0.004	0.9880±0.015

Table 5.3.4: Accuracy benchmark (RMSE) on plaintext

5.3.4 Hyper-parameter Setting

To train the CryptoRec's model, we perform a grid search for each hyper-parameter. In particular, for the learning rate η we search in $\{0.0001, 0.0002, 0.0004\}$; for the regular parameter λ we search in $\{0.00001, 0.00002, 0.00004\}$; for the dimension of the features $\{\mathbf{A}, \mathbf{Q}\}$, we search in $\{300, 400, 500, 600\}$. As a result, we choose $\eta = 0.0002$, $\lambda = 0.00002$ and the dimension $d = 500$. To train the baseline models, we also perform a grid search around the suggested settings given in their original papers, as the dataset splitting is not the same. By doing so, we have a fair comparison.

5.4 Performance Evaluation

In this section, we first evaluate and compare the accuracy and efficiency performance of using only a pre-trained model (Subsection 5.4.1). Then we investigate the accuracy and efficiency performance of a re-training process (Subsection 5.4.2).

5.4.1 Comparison with Pre-trained Models

As described in Section 5.3.2, CryptoRec's model, I-NBM, and BiasedMF allow computing recommendation with a pre-trained model, and the Client's private data is not in their training set. In this section, we first verify and compare the accuracy performance by directly using the pre-trained models (without re-training the models with the Client's data). Then we analyze and compare the computational complexity of responding one prediction query in a private manner. Compared to the complexity of homomorphic operations, algebraic operations in the plaintext space are trivial. As such, for the computational complexity analysis, we only count in the operations over encrypted data, i.e., operations between two ciphertexts (i.e., $\oplus$, $\otimes$) and multiplicative operations between a plaintext and a ciphertext (i.e., $\odot$).

5.4.1.1 Accuracy Comparison

Table 5.3.3 presents the accuracy performance of each model. Compared to the benchmark (Table 5.3.4), the accuracy of the three models is compromised to some extent (column loss%), and CryptoRec has the least loss. Specifically, CryptoRec loses 3.0% accuracy on netflix, 4.9 % on ml1m, and 0.1% on yahoo. Clearly, CryptoRec is able to provide a promising accuracy guarantee to the Client by using only a pre-trained model.

	$\oplus$	$\odot$	Sigmoid
I-NBM	$\mathcal{O}(m^2)$	$\mathcal{O}(m^2)$	$\varnothing$
U-AutoRec	$\mathcal{O}(md)$	$\mathcal{O}(md)$	$\mathcal{O}(md)$
CryptoRec	$\mathcal{O}(md)$	$\mathcal{O}(md)$	$\varnothing$

Table 5.4.1: Computational complexity comparison of using pre-trained models.

5.4.1.2 Computational Complexity Comparison

To respond to a query from the Client, the Server has to predict the Client's preferences on all the items since it gets only an encrypted rating vector ($[\![\mathbf{r}_u]\!]$). We analyze each model's computational complexity of answering one query, as shown Table 5.4.1. Among the three models, I-NBM consumes more homomorphic additions ($\oplus$) and multiplications ($\odot$); U-AutoRec costs a similar number of $\oplus$ and $\otimes$ than CryptoRec's model, but it introduces $\mathcal{O}(md)$ non-linear transformations (i.e., Sigmoid). Computing the Sigmoid function often relies on secure multiparty computation (SMC) schemes or polynomial-approximation [43, 72]. The former requires the Server and Client to be online constantly and pay the price of extra communication overhead [14, 72]; the latter leads to the use of a (somewhat) fully homomorphic encryption scheme since it introduces homomorphic multiplications between two ciphertexts ($\otimes$) [43]. Apparently, CrytoRec yields the best efficiency performance.

5.4.1.3 Evaluation of CryptoRec

As shown in Table 5.4.1, CryptoRec needs only homomorphic additions $\oplus$, and multiplications between ciphertexts and plaintexts $\odot$. As such, any additively hommorphic encryption can be employed to implement CryptoRec. In this experiment, we adopt the Paillier cryptosystem [91] implemented in the library python-paillier [10]. In the implementation, we scale-up the parameter values to integers, the Client can obtain correct recommendations by simply sorting the prediction results. We let secret key size $l = 2048$. In this setting, the message size of one encrypted rating $[\![r_{ui}]\!]$ is around 512 bytes, or 0.5 KB.

Following the pruning method proposed by [49], we remove the model parameters of which the values are very close to zero (i.e., $[-5 \times 10^{-4}, 5 \times 10^{-4}]$), since these model parameters don't contribute to the final predictions. Then we quantify the values of the left model parameters to be 11 bits (2048 shared parameter values), of which we can reuse most of the related computations. It is worth mentioning that this approach does not compromise the accuracy, sometimes, it even leads to a slightly better accuracy performance. The same phenomenon has been also observed by some other works such as [49, 88].

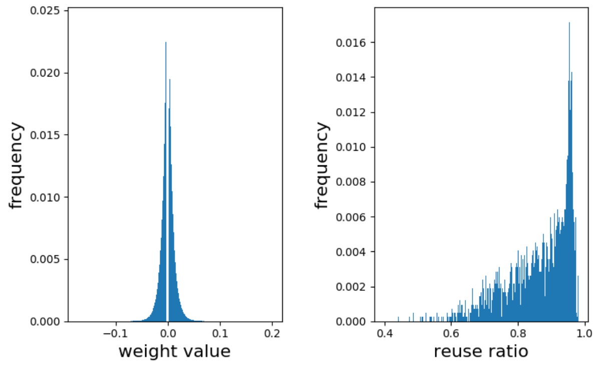


Figure 5.4.1: CryptoRec's model trained on dataset ml1m: the distribution of parameter values after pruning (left panel); the distribution of the reuse ratio of each row of model parameters **A** and $\mathbf{Q}^T$ after the quantification (right panel).

The left panel of Figure 5.4.1 intuitively describes the model parameter values distribution after the pruning, and the right panel of Figure 5.4.1 is the reuse ratio distribution of each row of model parameters **A** and $\mathbf{Q}^T$ after the quantification, the model here is trained on dataset ml1m. Table 5.4.2 presents the **pruning ratio** and overall **reuse ratio** of the CryptoRec's model trained on each dataset, where we define the **pruning ratio** as $\frac{\#\ \textit{of pruned parameter}}{\#\ \textit{of all the parameter}}$, and compute the **reuse ratio** as $1 - \frac{\#\ \textit{of unique parameter}}{\#\ \textit{of all parameter}}$.

According to Table 5.4.2, we know that reusing computations on the shared parameter values is able to significantly reduce the computational complexity. For

	netflix	ml1m	yahoo
pruning ratio	7.1%	9.2%	29.4%
reuse ratio	90.7%	90.5%	91.5%

Table 5.4.2: Parameter pruning ratio and computation reuse ratio of CryptoRec's models

example, when computing $[\![r_{ui}]\!] \odot \mathbf{A}_{j:} = \{[\![r_{ui}]\!] \odot \mathbf{A}_{j1}, [\![r_{ui}]\!] \odot \mathbf{A}_{j2}, \cdots, [\![r_{ui}]\!] \odot \mathbf{A}_{jd}\}$, we only need to compute $\odot$ operations on each shared parameter value of $\mathbf{A}_{j:}$ (j-th row of $\mathbf{A}$), and then reuse the results at the other places of $\mathbf{A}_{j:}$.

	netflix	ml1m	yahoo
Communication (MB)	4.8	3.86	3.72
Server time cost (s)	14.2	10.9	7.3
Client time cost (s)	7.1	5.8	5.6

Table 5.4.3: The communication (MB) and time (s) cost of CryptoRec with a pre-trained model

We summarize the communication and time cost of the Client and Server in Table 5.4.3. To elaborate the prediction process and the costs, we take the experiment on dataset ml1m as an example (We ignore the time cost of a public key pair generation, as it is trivial to the overall time cost),

- Client: Encrypting the rating vector $[\![\mathbf{r}_u^{1\times 3952}]\!]$ takes 4.5 seconds. The message size of $[\![\mathbf{r}_u^{1\times 3952}]\!]$ is 0.5×3952 KB, or 1.93 MB.
- Server: Executing CryptoRec on $[\![\mathbf{r}_u^{1\times 3952}]\!]$ takes 10.9 seconds. The message size of the output $[\![\hat{\mathbf{r}}_u^{1\times 3952}]\!]$ is 1.93 MB.
- Client: Decrypting $[\![\hat{\mathbf{r}}_u^{1\times 3952}]\!]$ takes 1.3 seconds.

We also implement the prediction process of I-NBM with the Paillier cryptosystem, where the item-item similarity matrix is pre-computed. Selecting the most similar N items to a targeted item from a user's rating history is a typical approach used in I-NBM to compute recommendations. However, this approach introduces

	netflix			ml1m			yahoo		
	RMSE	loss%	iter#	RMSE	loss%	iter#	RMSE	loss%	iter#
I-NBM	0.9061±0.005	8.7	1	0.8815±0.007	5.4	1	0.9853±0.014	-0.3	1
U-AutoRec	0.8849±0.007	6.2	35	0.8739±0.009	4.4	30	1.0583±0.016	7.1	26
I-AutoRec	*0.8334±0.006*	*0*	*140*	*0.8367±0.004*	*0*	*110*	*0.9880±0.015*	0	*125*
BiasedMF	0.8587±0.007	3.0	85	0.8628±0.009	3.1	80	0.9980±0.022	1.0	72
CryptoRec	**0.8391±0.006**	**0.7**	**12**	**0.8543±0.007**	**2.1**	**15**	**0.9821±0.013**	**-0.6**	**22**

Table 5.4.4: Accuracy comparsion with a re-training step, I-AutoRec is the accuracy benchmark.

a number of extra non-linear operations (i.e., comparisons) which are not straightforwardly compatible with homomorphic encryption schemes. To address this issue, for each entry of the similarity matrix, we remove a certain number (e.g., 30%) of elements which have the least values. The predictions computed on the sparsified similarity matrix are asymptotically close to the true predictions. In fact, using all the items for the prediction may lead to a significant accuracy loss. In our implementation, for one query, I-NBM requires 491 seconds, 335 seconds and 306 seconds on netflix, ml1m, yahoo datasets, respectively. We noted that Shmueli at al. [107] used an additional mediator (i.e., a non-colluding global server) to achieve a more efficient solution. However, we focus on the 2PC protocol without using any third party, and in their setting, participants know which item to predict while in our case the Server doesn't know it. It is not necessary to include U-AutoRec in the comparison, because Sigmoid transformations it contains will result in a much worse efficiency performance.

CryptoRec's model allows providing accurate recommendations by a pre-trained model (the Client's data is not in the training set). So, the Server can provide recommendation services with a high throughput. In contrast, for the models which fall into category "w/ Client", the time cost of the training process should be also counted, which leads to a notorious efficiency problem. For example, privately training matrix factorization on the dataset ml1m needs around 20 hours per iteration (more details are in Section 5.4.2.4).

	$\oplus$	$\otimes$	$\odot$	div	Sigmoid	sqrt
I-NBM	$\mathcal{O}(m)$	$\mathcal{O}(m^2)$	$\mathcal{O}(m)$	$\mathcal{O}(m^2)$	$\varnothing$	$\mathcal{O}(m)$
U-AutoRec	$\mathcal{O}(K(m+N_\tau)d)$	$\mathcal{O}(K(m+N_\tau)d)$	$\mathcal{O}(K\tau zd+md)$	$\varnothing$	$\mathcal{O}(Kmd)$	$\varnothing$
I-AutoRec	$\mathcal{O}(K(mn+N)d)$	$\mathcal{O}(K(m+N)d)$	$\mathcal{O}(Kmzd)$	$\varnothing$	$\mathcal{O}(Knd)$	$\varnothing$
BiasedMF	$\mathcal{O}(Kmnd)$	$\mathcal{O}(K(m+N)d)$	$\mathcal{O}(md)$	$\varnothing$	$\varnothing$	$\varnothing$
CryptoRec	$\mathcal{O}(K(m+N_\tau)d)$	$\mathcal{O}(K(m+N_\tau)d)$	$\mathcal{O}(K\tau zd+md)$	$\varnothing$	$\varnothing$	$\varnothing$

Table 5.4.5: Computational complexity comparison with a re-training step. K is the number of training iterations. N is the number of all observed ratings in the Server's dataset. N_τ is the number of observed ratings of the τ randomly selected users. z is the rating scale ($z = 5$ in this experiment).

5.4.2 Comparison with a Re-training Step

In this section, we investigate the accuracy and efficiency performances of using the Client's data to re-train a pre-learned CryptoRec's model. We first describe the details of re-training CryptoRec's model, then introduce a one-iteration training method for the sake of efficiency.

5.4.2.1 Re-training CryptoRec's Model

Avoiding Overfitting. Using a single user's data to fine-tune a machine learning model learned from a large dataset may lead to an early overfitting. To address this issue, we re-train CryptoRec's model using the Client's data together with τ randomly selected users' data, where the τ users serve as a regularization term. We empirically set $\tau = 10$, and $\tau \ll n$ (n is the number of users in the Server's dataset).

Stopping Criterion. Identifying the stopping point of a training process over encrypted data is not as straightforward as doing that on clear data. This is because the Server gets only an encrypted model, that the accuracy performance at each training iteration cannot be observed. To address this issue, the early-stopping strategy [99] can be a choice. Fortunately, we have also observed that, for the re-training process, the first several training iterations contribute most to the accuracy increase (RMSE decrease), as shown in Figure 5.4.2 and Table 5.4.6. Specifically, the first training iteration leads to a big step towards the optimal accuracy perfor-

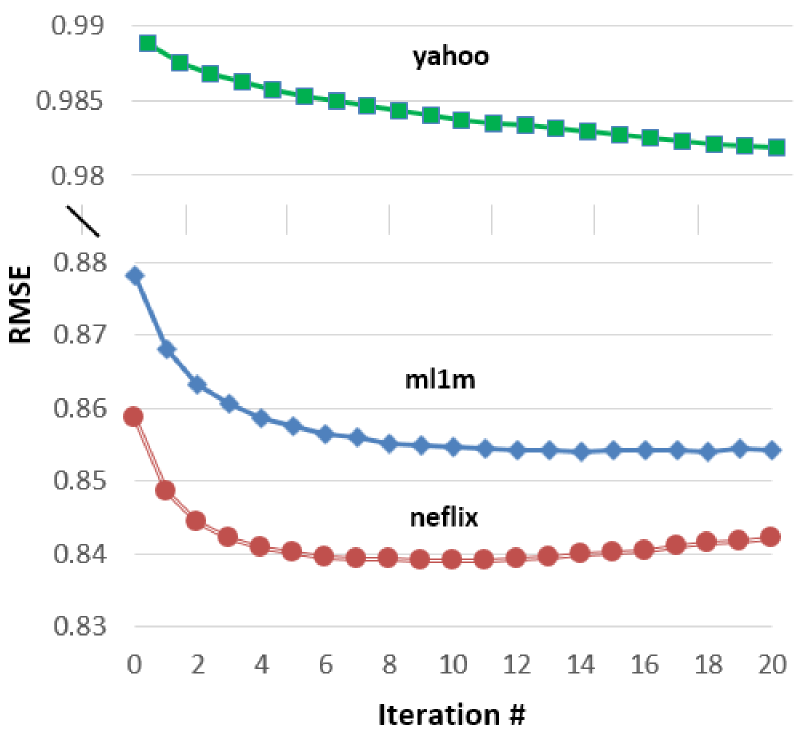


Figure 5.4.2: Re-training CryptoRec with different iteration#

mance. With 3 to 5 iterations, the accuracy performance can be asymptotically close to the best accuracy. Therefore, we can conservatively re-train CryptoRec's model (e.g., 4 iterations) while still leading to a nearly consistent accuracy optimization.

5.4.2.2 Accuracy comparison

We summarize the accuracy performance of each model in Table 5.4.4, experiment results show that the accuracy performance of CryptoRec is competitive with the benchmark (as described in Table 5.3.4) and consistently outperforms the other baseline models. Specifically, compared to the benchmark, CryptoRec loses 0.7% accuracy on netflix; loses 2.1% accuracy on ml1m; on yahoo, CryptoRec slightly outperforms the benchmark. Note that Table 5.4.4 presents the optimal accuracy

	w/o retrain	retrain-full		retrain-once	
	(RMSE)	RMSE	inc%	RMSE	inc%
netflix	0.8586	0.8391	2.3	0.8485	1.18
ml1m	0.8781	0.8543	2.7	0.8680	1.15
yahoo	0.9888	0.9821	0.7	0.9874	0.14

Table 5.4.6: CryptoRec accuracy comparison: without retraining (w/o retrain) - retrain until convergence (retrain-full) - retrain only once (retrain-once). "inc%" denotes the percentage of the accuracy increase.

performance of each model. In practice, the Server may achieve a suboptimal accuracy performance, due to the stopping point selection strategy or the constraint of computational resource. Roughly, predictions using only a pre-trained model reach the lower-bound of accuracy, the re-training process leads to a better accuracy performance. The optimal accuracy performance can guide users to perform the trade-off between efficiency and accuracy.

5.4.2.3 Computational Complexity Comparison

Table 5.4.5 presents the computational complexity of each model. Among all the models, only MF and CryptoRec's model can be trained without using non-linear operations. However, MF has to be trained on the whole dataset which results in a serious efficiency issue. In contrast, re-training CryptoRec needs only the Client's data and the data of several randomly selected users (for regularization). We have noted that some researchers proposed incremental matrix factorization training methods such as [53, 111, 120]. Unfortunately, these incremental training methods either require the Server to collect partial data of the Client [53, 120] (we assume that the Server has no prior knowledge of the Client' rating data), or introduce extra non-linear operations [111]. Therefore, we don't include these incremental matrix factorization training methods in the comparison. As presented in Table 5.4.5, CryptoRec shows a significant advantage in the efficiency performance.

5.4.2.4 Evaluation of CryptoRec

Re-training CryptoRec's model needs a (somewhat) fully homomorphic encryption scheme (SWHE) since homomorphic addition ($\oplus$) and multiplication ($\otimes$) are both required. Some of the more significant advances in implementation improvements for SWHEs have come in the context of the ring-learning-with-error (RLWE) based schemes, such as Fan-Vercauteren scheme [37]. RLWE-based homomorphic encryption schemes map a plaintext message from ring $\mathfrak{R}_t^p := \mathbb{Z}_t[x]/(x^p + 1)$ to ring $\mathfrak{R}_t^p := \mathbb{Z}_q[x]/(x^p + 1)$ (ciphertext). The security level depends on the plaintext modulus t, the coefficient modulus q, the degree p of the polynomial modulus. In this implementation, we adopt the Fan-Vercauteren scheme, a real-number-supported version of which is implemented in the SEAL library [22]. We set the polynomial degree $p = 4096$, the plaintext modulus $t = 65537$, q is automatically selected by the SEAL library given the degree p. To encode real numbers, we reserve 1024 coefficients of the polynomial for the integral part (low-degree terms) and expand the fractional part to 16 digits of precision (high-degree terms). The circuit privacy is guaranteed by using *relinearization* operations [22, Section 8]. We refer interested readers to the paper [22] for more detail of the settings.

Compared to partial homomorphic encryption schemes such as the Paillier cryptosystem, using an SWHE scheme results in a much larger ciphertext, which in turn leads to a higher computational complexity for a homomorphic operation. In this experiment, the polynomial degree $p = 4096$. Each coefficient of the polynomial costs 24 bytes (using SEAL) [43]. So the size of a ciphertext is 4096*24 bytes or 96 KB. Taking the re-training process on dataset ml1m as an example, the item features $[\![\mathbf{Q}^{3952\times 500}]\!]$ need 3952*500*96 KB or 181 GB RAM. Though it is not infeasible for a commercial server, it is too expensive to respond to a single query while the accuracy improvement is limited.

By exploiting the fact that the first re-training iteration contributes a big portion to the accuracy increase (Table 5.4.6), we introduce an efficient one-iteration re-training method, described in Algorithm 6[3]. The gradients of parameters are

[3]For simplicity, we omitted from Algorithm 6 the bias terms and the τ number of randomly

Algorithm 6 Re-train CryptoRec's model with one iteration

```
1: procedure RE-TRAIN($[\![\mathbf{r}_u]\!], [\![\varphi_u]\!], \mathbf{A}^{(0)}, \mathbf{Q}^{(0)}, \lambda, \eta$)
2:     $[\![\mathbf{y}_u]\!] \leftarrow [\![\mathbf{r}_u]\!]\mathbf{A}^{(0)}$
3:     $[\![\mathbf{e}_u]\!] \leftarrow [\![\hat{\mathbf{r}}_u]\!] \ominus [\![\mathbf{r}_u]\!] = [\![\mathbf{y}_u]\!]\mathbf{Q}^{(0)} \ominus [\![\mathbf{r}_u]\!]$
4:     $[\![\mathbf{x}_u]\!] \leftarrow ([\![\mathbf{e}_u]\!] \otimes [\![\varphi_u]\!])\mathbf{Q}^{(0)}$
5:     for $j \leftarrow \{1, 2, \cdots, d\}$ do
6:         $[\![\Delta\mathbf{A}_{:j}]\!] \leftarrow ([\![\mathbf{x}_u]\!][j] \otimes [\![\mathbf{r}_u^T]\!]) \oplus \lambda \cdot \mathbf{A}_{:j}^{(0)}$    ▷ gradient
7:         $[\![\mathbf{A}_{:j}]\!] \leftarrow \mathbf{A}_{:j}^{(0)} \ominus (\eta \odot [\![\Delta\mathbf{A}_{:j}]\!])$    ▷ updates $\mathbf{A}_{:j}$
8:         $[\![\mathbf{p}_u]\!][j] \leftarrow [\![\mathbf{r}_u]\!][\![\mathbf{A}_{:j}]\!]$    ▷ computes user features
9:         release $[\![\mathbf{A}_{:j}]\!]$, $[\![\mathbf{x}_u]\!][j]$
10:    release $[\![\mathbf{r}_u]\!]$
11:    for $i \leftarrow \{1, 2, \cdots, m\}$ do
12:        $[\![\Delta\mathbf{q}_i]\!] \leftarrow [\![\varphi_u]\!][i] \otimes (([\![\mathbf{e}_u]\!][i] \otimes [\![\mathbf{y}_u]\!]) \oplus \lambda \cdot \mathbf{q}_i^{(0)})$
13:        $[\![\mathbf{q}_i]\!] \leftarrow \mathbf{q}_i^{(0)} \ominus (\eta \odot [\![\Delta\mathbf{q}_i]\!])$    ▷ updates $\mathbf{q}_i$
14:        $[\![\hat{\mathbf{r}}_u]\!][i] \leftarrow [\![\mathbf{p}_u]\!][\![\mathbf{q}_i]\!]$    ▷ computes the prediction $\hat{r}_{ui}$
15:        release $[\![\mathbf{q}_i]\!]$, $[\![\mathbf{e}_u]\!][i]$, $[\![\varphi_u]\!][i]$
16:    return $[\![\hat{\mathbf{r}}_u]\!]$
```

presented in Equation (5.8). The basic idea of this method is to timely release the model parameters which will not be used in the future (line 9, 10, 15, Algorithm 6). For example, we immediately release $[\![\mathbf{A}_{:j}]\!]$ and $[\![\mathbf{x}_u]\!][j]$ after computing $[\![\mathbf{p}_u]\!][j]$ (line 9), where $\mathbf{A}_{:j}$ denotes j-th column of matrix $\mathbf{A}$. $\mathbf{Q}^{(0)}$ and $\mathbf{A}^{(0)}$ are pre-trained model parameters. $[\![\mathbf{x}]\!] * [\![\mathbf{y}]\!]$ denotes $\{[\![x_i]\!] * [\![y_i]\!]\}_m$ and $x * [\![\mathbf{y}]\!]$ denotes $\{x_i * [\![y_i]\!]\}_m$, where $*$ can be any operator such as $\oplus$, $\otimes$. $\ominus$ is homomorphic subtraction which can be implemented by $\oplus$. With Algorithm 6, we can complete the one-iteration training process (including computing the predictions) with less than 2 GB RAM.

We summarize the communication and time costs of the Client and Server in Table 5.4.7. We take the experiment on dataset ml1m as an example to introduce the cost on the two sides, respectively,

chosen users (Section 5.4.2.1). Note that the gradients computed from the data of the τ users and the pre-trained model parameters are plaintext. Therefore, all the operations related to the τ users are in plaintext, and have a trivial impact on the efficiency performance.

	netflix	ml1m	yahoo
Communication (GB)	1.31	1.08	1.04
Server time cost (H)	9.4	7.8	7.5
Client time cost (s)	14.3	11.8	11.4

Table 5.4.7: The communication and time cost of CryptoRec with one-iteration re-training process

- Client: Encrypting the rating vector $\mathbf{r}_u^{1\times 3952}$ and indication vector $\varphi_u^{1\times 3952}$ takes 9.6 seconds. The message size of $[\![\mathbf{r}_u^{1\times 3952}]\!]$ and $[\![\varphi_u^{1\times 3952}]\!]$ is $96 \times 3952 \times 2$ KB, or 741 MB.

- Server: Executing CryptoRec on $[\![\mathbf{r}_u^{1\times 3952}]\!]$ takes 7.8 hours. The message size of the output is $[\![\hat{\mathbf{r}}_u^{1\times 3952}]\!]$ is 370.5 MB.

- Client: Decrypting $[\![\hat{\mathbf{r}}_u^{1\times 3952}]\!]$ takes 2.2 seconds.

In contrast, the models which fall into "w/ Client" category lead to a much higher time cost. For example, a recent work, GraphSC [83], shows that a single iteration of training MF (the dimension of user/item features is 10), on the same dataset ml1m, took roughly 13 hours to run on 7 machines with 128 processors. In our setting, by making full use of the fact that the Server knows most of the users data, it still needs around 20 hours for any i-th iteration with 8 processors, where $i > 1$. Worse, dozens of iterations are necessary for convergence [64, 86].

5.4.3 Discussion on Privacy and Scalability

We assume that the Server and Client should always agree upon a set of items, as it does not make sense to buy a service that the other party doesn't have, and vice versa. In fact, this assumption leads to a a trade-off between privacy and scalability. Informally, the more items that the two sides agreed on, the more privacy can be preserved. An online service provider (e.g., Youtube) may have millions of products, it is a notoriously challenging problem to provide recommendations from such a large corpus, even on clear data. A typical approach is to generate a small set of candidates, then compute recommendations from the candidates [24]. For our scenario, context information can be used to guide candidate generation, but still depending on whether such a context information is a privacy issue that the Client cares about (different users may have different concerns about privacy). The Server can train different recommendation models over datasets generated by different criteria such as children-friendly, place-of-origin, time-of-produce and so on. The Client can choose a criterion which doesn't violate her privacy concern, or choose multiple criteria at a time. How to design these criteria requires a further investigation of user preferences on privacy.

5.5 Knowledge Transferring

Knowledge transferring, a.k.a transfer learning [92], has recently attracted many attentions from machine learning community [19]. In recommender systems, it is a challenging and still largely under-explored field [19]. For example, the existing researches focus on cold start problem and generation of recommendations from multiple domains [19, 68, 93]. But the efficiency performance, which is crucial to the applications on encrypted data, is rarely taken into their consideration. As a byproduct, thanks to its user-free latent feature space, CryptoRec's model naturally achieves transferability. As such, it can easily transfer the knowledge learned from one database (source) to another (target) if they share the same item set. This transferability facilitates both efficiency and accuracy performance, and it also provides a concrete evidence that the Server is able to provide reliable recommendation services to diversified customers, e.g., the customers from different e-commercial websites or who bought products from different shops. We will also discuss how the transferability can facilitate privacy-protection.

source	target	no transfer learning		transfer learning	
		RMSE	iteration#	RMSE	iteration#
ml1m*$_1$	yahoo*	0.8664±0.024	122	0.8312±0.016	32
yahoo*	ml1m*$_1$	0.8460±0.008	120	0.8463±0.011	55
neflix*	ml1m*$_2$	0.8527±0.003	118	0.8421±0.004	26
ml1m*$_2$	neflix*	0.8005±0.004	95	0.7981±0.005	28

Table 5.5.2: Transfer learning helps both accuracy (RMSE) and efficiency (# of iterations)

item intersection	data set	user#	item#	density
ml1m $\bigcap$ yahoo	ml1m*$_1$	6040	2715	5.6%
	yahoo*	5507	2715	0.5%
ml1m $\bigcap$ neflix	ml1m*$_2$	6040	2718	5.1%
	netflix*	10000	2718	9.2%

Table 5.5.1: New datasets resulting from the intersections other datasets

To evaluate the transferability of CryptoRec, we have constructed four new datasets, Table 5.5.1, where ml1m*$_1$ and yahoo* are two item-entry shared datasets which are created from the datasets m1lm and yahoo, respectively; and ml1m*$_2$ and netflix* are another two item-entry shared datasets which are extracted from the datasets m1lm and netflix, respectively.

The knowledge transferring process is simple and straightforward. Firstly, we pre-train CryptoRec' model on one dataset (e.g., ml1m*$_1$), and then use the pre-trained CryptoRec model to initialize the training of CryptoRec's model on another dataset (e.g., yahoo*). The accuracy performance is presented in Table 5.5.2. The results show that using the knowledge learned from a relatively dense dataset, we can improve both the accuracy and efficiency performance of CryptoRec's model on a relatively sparse dataset. For example, the density of dataset ml1m*$_1$ and yahoo* are 5.6% and 0.5% respectively. We take dataset ml1m$*_1$ as the source and yahoo* as the target. By using the knowledge learned from ml1m*$_1$, CryptoRec's model achieves 4.1% accuracy increase, and the number of training iterations is also significantly reduced (from 122 to 32). Transfering the knowledge learned from a relatively sparse dataset does not always increase the accuracy performance of CryptoRec's model on a relatively dense dataset, but it consistently reduces the number of training iterations. We again take datasets ml1m$*_1$ and yahoo* as example, the accuracy performance is slightly different, but the number of training iterations is reduced from 120 to 55. The two observations are not counter-intuitive. The optimization problem, i.e., minimizing the objective function (e.g., Equation (5.7)), is often non-convex. So a good initialization for the model parameters Θ can facilitate the searching of a better local optimum [36]. A relatively dense dataset may contain more information than a relatively sparse dataset, so using the knowledge learned from this dataset as the source can improve both efficiency and accuracy. Using the knowledge learned from a relatively sparse dataset as the source may also result in a good initialization, thus to improve the efficiency performance.

Privacy-preserving solutions can also benefit from the reduction of training iterations. For users who hold some data and want to jointly train a model to obtain recommendation service, they can collect some public data to pre-train CryptoRec's model. No matter what privacy-preserving frameworks they use, e.g., [80, 86, 108], reducing the training iterations will lead to an immediate efficiency performance improvement, e.g., lower communication overhead and computation cost. For a server who wants to provide differential privacy guarantee [33] to its recommender system, this transferability can facilitate a better trade-off between accuracy and privacy. For example, iteratively training a model accumulates the privacy loss, which is theoretically discussed by the composition theory of differential privacy ([33, Section 3]). So reducing the number of training iterations immediately reduces the privacy loss.

5.6 Related Work

Some recent works, such as [43, 72, 103], focused on neural network based Machine Learning as a Service (MLaaS), the scenario of which is similar to ours. Their primary contribution is how to efficiently compute non-linear operations (e.g., comparison or Sigmoid function) on encrypted data. Gilad-Bachrach et al. [43] substituted state-of-the-art activation functions such as ReLu ($relu(x) = max(0, x)$) with a simple square activation function ($f(x) = x^2$), this avoid the use of secure multiparty computation schemes. However, this approach often leads to a significant accuracy loss [72, 103]. To preserve the accuracy performance, Liu et al. [72] and Rouhani et al. [103] proposed to evaluate neural networks with resort to secure multiparty computation schemes. Unfortunately, this approach requires the Client and Server to be online constantly. State-of-the-art recommendation algorithms often require jointly learning personalized user features and item features. It means that we cannot assume a pre-trained model to compute recommendations based on existing recommendation algorithms. Therefore, we cannot directly applying the above solutions to Recommendation as a Service (RaaS).

There are some solutions allowing securely training machine learning models. For example, Canny et al. [18] introduced a privacy-preserving solution for training collaborative filtering models (e.g., Singular Value Decomposition) in a peer-to-peer manner without assuming any trusted server. Nikolaenko et al. [86] proposed a garbled circuits [62] based secure protocol to allow multiple users jointly train matrix factorization (MF), in which they assume two non-colluding servers. Shmueli et al. [107] discussed that multi-party privately learn a neighborhood-based recommendation model by assuming a mediator that performs intermediate computations on encrypted data supplied by each party. Nayak at al. [83] brought parallelism to the secure implementation of oblivious version of graph-based algorithms (e.g., MF). Mohassel et al. [80] further improved the efficiency of a secure framework with two non-colluding servers. Different from these solutions, we aim to build a secure two-party computation protocol for RaaS, without involving any third party (e.g., an additional non-colluding server).

An orthogonal line of work focuses on constructing differentially private machine learning models, e.g., [73, 76, 108]. In their security models, a trusted server has full access to all the user data. It wishes to prevent adversaries from breaching the user privacy by exploiting the prediction results (i.e., inference attack). In our security model, the Server learns nothing about client inputs; at the same time, the

Client only learns what she can learn from the recommendation results. Our work and differential privacy [33] can be complementary to each other.

5.7 Conclusion and Future Work

With the rising interest in machine learning from both academia and industry, providing recommendations as a service has great potential impact in the years to come. However, confidentiality concerns have to be considered.

In this chapter, we proposed CryptoRec: a two-party protocol where a server with large dataset provides recommendations to a client, based on the client's preferences. Our protocol encompasses a crypto-friendly recommender system that is able to produce recommendations without having to re-train the model with the client's preferences. Together with homomorphic encryption, we obtain a secure protocol where the server learns nothing about the client's preferences, and the client learns only the computed recommendation and nothing about the server's pre-computed model parameters (beyond, obviously, what the client can infer from the returned recommendation itself).

The efficiency of CryptoRec results from two remarkable properties of its underlying recommender system that: (1) requires only addition and multiplication operations, making it straightforwardly compatible with homomorphic encryption schemes; (2) computes recommendations without having to re-train the model parameters with the client's preferences.

Using standard metrics (RMSE), benchmarks on public datasets of movies ratings show that CryptoRec suffer from less than 5% accuracy loss when compared with the state-of-the-art recommender system computing over clear data. This accuracy loss can be further reduced if we allow the server to re-train the model parameters with the client's input, but the computational cost of computing a recommendation becomes considerable for regular computers. Further improvements on the prediction accuracy without compromising security and without significantly affecting the efficiency of the protocol may require new ideas and techniques.

6 Summary

6.1 Conclusions

State-of-the-art recommender systems directly capture user preferences from historical user behaviours (e.g., user-location, user-product, etc), and a clear tendency is that more and more user data will be adopted to construct new recommender systems [3, 132]. This fact would result in more privacy risks to users, which in turn may lead to a crisis of trust to recommendation services. Naturally, privacy-preserving recommendation services (and also other machine learning services) becomes a necessary and serious topic. A remarkable progress has been achieved for privacy-preserving in the last ten years. At the meanwhile, it also has formed a pattern that machine learning and privacy-preserving are two competing goals at stake. In this dissertation, we explored the possibility of facilitating privacy-preserving recommender systems from the machine learning side. We show that, in the setting of privacy-preserving, machine learning can be thought of as an ally rather than a foe.

We firstly investigated privacy-preserving recommender systems under the centralized case. In this case, A trusted server provides recommendation services to the users whose data that it can fully access. The users obtain the recommendation results, while the privacy loss of any of them is strictly controlled. In particular, we proposed a probabilistic neighborhood-based method, which enables a flexible approach to apply differential privacy to neighborhood-based methods, either by adding carefully calibrated noise into each training step or by sampling from the scaled posterior distribution over similarities between any two items. The experiment results shows that our proposed privacy-preserving solutions allows a priming accuracy performance with a meaningful differential privacy guarantee.

Then, we investigated privacy-preserving recommender systems under the distributed case. In this case, multiple data owners wish to jointly perform a machine learning task without leaking any of their private data. We looked into two directions, the first one is how to efficiently computation with social connections; the second one is how to build crypto-friendly recommender systems. In the first direction, we presented a social-context based recommender system. It allows computing accurate recommendations based on a few users who share certain social connections. We implemented a secure protocol with this system, with resort to a somewhat (fully) homomorphic encryption scheme. This recommender system requires a much fewer number of users (for recommendation computations) than existing recommendation algorithms do, significantly improving the efficiency performance. In the second direction, we proposed an HE-friendly recommender system. This recommendation model possesses two important properties: (1) It uses only addition and multiplication operations, so that it is straightforwardly compatible with HE schemes. With this property, CryptoRec is able to complete recommendation computations without requiring the Server and the Client to be online continuously; (2) It can automatically extract personalized user representations by aggregating pre-learned item features. This property allows the Server with a pre-trained model to provide recommendation services without a tedious re-training process, which significantly improves the efficiency performance. Note that the Client's data is not in the Server's database which is used for model training. Experiment results show that the proposed model allows the Server with thousands of items to privately answer a prediction query in a few seconds on a single PC.

In summary, this dissertation aims to use machine learning to improve the privacy-preserving solutions for recommender systems. We developed different solutions according to different security models. The results of each proposed solution demonstrated that privacy-protection and machine learning can be two allies rather than two competing boxers.

6.2 Future Works

Privacy and utility (e.g., efficiency and accuracy) have been long-term considered as two competing goals at stake. Though designing sophisticated secure protocols or improving cryptographic primitives is a typical approach and attracting a lot of attentions, it is still far from the satisfactory of practice. Our future work will follow the direction of using machine learning to solve the privacy issues of machine learning, as we believe that privacy and utility can be two allies in the same trench.

- *Privacy Transfer.* Iteratively training a machine learning model may lead to an additive increase of differential privacy loss, which is a bottleneck of that building differential privacy into state-of-the-art machine learning algorithms. Reducing the number of training iterations directly reduces the privacy loss. However, straightforwardly reducing the training iterations may result in a significant accuracy loss. In one of our work, we demonstrated that the number of training iterations (on the targeted database) can be significantly reduced without losing any accuracy, by transferring the knowledge from a (public) source database, where we assume that the two databases share the same items-entries. In practice, the requirement of two databases sharing the same item-entries is too strict. Therefore, it will be an exciting topic to design a new knowledge transferable recommender system which allows transferring knowledge from a source database which doesn't share the same item-entries with the target database. Note that different from existing cross-domain recommender systems, the training process of this new recommender system have to converge with a much fewer training iterations, by using the knowledge transferred from other databases.

- *Privacy Incentive.* Profits incentive commercial companies to be aggressive on various kinds of user data, such as locations, browsing history and even call records, providing high-quality recommendation services. Privacy has been long-time treated as a competing goal to the commercial profits, which discourages investments in privacy-preserving. The overuse of personal data may also devastate the user experience of recommendation services. To address this issue, a new recommendation algorithm with a privacy-incentive mechanism can be a promising direction. The key idea is to identify the contribution of each input (a kind of user data) to the preference estimation of an item, or which inputs result in an antipathy to recommended items. This means that the recommender system can estimate the privacy concern of a user, intelligently avoiding collecting or using the data conflicted with the user's privacy concern. By this, the system can improve user experience while mitigating privacy loss, thus to incentive commercial companies to preserve user privacy. Attention mechanism [128] is a possible solution to design such a privacy-incentive recommender system, deserving a further investigation.

- *Next Generation (encryption-aware) Recommender Systems.* Traditional recommendation methods, such as neighborhood-based methods and matrix factorization, fall short in expressiveness. They are also difficult in jointly modeling various inputs for estimating user preferences. It is foreseeable that more complex algorithms, such as deep models with non-linear transformations, will be used to construct next generation recommender systems. More user data may also be required to feed the new models. Simply adopting existing privacy-preserving solutions, or applying cryptographic primitives to these complicated recommender systems, would result in an efficiency bottleneck. Designing new algorithms which have a competitive performance with state-of-the-art recommender systems while still friendly to encrypted data, can be a direction deserves more effort to work on.

References

[1] Yahoo ! R4 - yahoo! movies. URL https://webscope.sandbox.yahoo.com/.

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016.

[3] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.

[4] Sungjin Ahn, Anoop Korattikara, Nathan Liu, Suju Rajan, and Max Welling. Large-scale distributed bayesian matrix factorization using stochastic gradient mcmc. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 9–18. ACM, 2015.

[5] E. Aïmeur, G. Brassard, J. M. Fernandez, and F. S. M. Onana. Alambic: a privacy-preserving recommender system for electronic commerce. *Int. J. Inf. Secur.*, 7:307–334, 2008.

[6] amazon. Real-time product recommendations. URL https://aws.amazon.com/mp/scenarios/bi/recommendation/.

[7] Roberto J Bayardo and Rakesh Agrawal. Data privacy through optimal k-anonymization. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 217–228. IEEE, 2005.

[8] Amos Beimel, Hai Brenner, Shiva Prasad Kasiviswanathan, and Kobbi Nissim. Bounds on the sample complexity for private learning and private data release. *Machine learning*, 94(3):401–437, 2014.

[9] Arnaud Berlioz, Arik Friedman, Mohamed Ali Kaafar, Roksana Boreli, and Shlomo Berkovsky. Applying differential privacy to matrix factorization. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 107–114. ACM, 2015.

[10] John Bethencourt. Paillier library. URL `http://acsc.cs.utexas.edu/libpaillier/`.

[11] R. Bhaskar, A. Bhowmick, V. Goyal, S. Laxman, and A. Thakurta. Noiseless database privacy. In D. H. Lee and X. Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 215–232. Springer, 2011.

[12] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191. ACM, 2017.

[13] Joppe W Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *Cryptography and Coding*, pages 45–64. Springer, 2013.

[14] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS*, 2015.

[15] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):13, 2014.

[16] Joseph A Calandrino, Ann Kilzer, Arvind Narayanan, Edward W Felten, and Vitaly Shmatikov. " you might also like:" privacy risks of collaborative filtering. In *2011 IEEE Symposium on Security and Privacy*, pages 231–246. IEEE, 2011.

[17] John Canny. Collaborative filtering with privacy via factor analysis. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '02, pages 238–245, New York, NY, USA, 2002. ACM. ISBN 1-58113-561-0. doi: 10.1145/564376.564419. URL `http://doi.acm.org.proxy.bnl.lu/10.1145/564376.564419`.

[18] John Canny. Collaborative filtering with privacy. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 45–57. IEEE, 2002.

[19] Iván Cantador, Ignacio Fernández-Tobías, Shlomo Berkovsky, and Paolo Cremonesi. Cross-domain recommender systems. In *Recommender Systems Handbook*, pages 919–959. Springer, 2015.

[20] Kamalika Chaudhuri and Nina Mishra. When random sampling preserves privacy. In *Annual International Cryptology Conference*, pages 198–213. Springer, 2006.

[21] Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. In *Advances in Neural Information Processing Systems*, pages 289–296, 2009.

[22] Hao Chen, Kim Laine, and Rachel Player. Simple encrypted arithmetic library-seal v2. 1. In *International Conference on Financial Cryptography and Data Security*, pages 3–18. Springer, 2017.

[23] Tianqi Chen, Emily B Fox, and Carlos Guestrin. Stochastic gradient hamiltonian monte carlo. In *ICML*, pages 1683–1691, 2014.

[24] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 191–198. ACM, 2016.

[25] Christian Desrosiers and George Karypis. A comprehensive survey of neighborhood-based recommendation methods. *Recommender systems handbook*, pages 107–144, 2011.

[26] Irit Dinur and Kobbi Nissim. Revealing information while preserving privacy. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 202–210. ACM, 2003.

[27] S. Dooms, T. De Pessemier, and L. Martens. Movietweetings: a movie rating dataset collected from twitter. In *Proceedings of Workshop on Crowdsourcing and Human Computation for Recommender Systems*, pages 84–89, 2013.

[28] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing. https://www.microsoft.com/en-us/research/publication/manual-for-using-homomorphic-encryption-for-bioinformatics/.

[29] Y. Duan. Privacy without noise. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 1517–1520. ACM, 2009.

[30] Yitao Duan and John F Canny. Practical private computation and zero-knowledge tools for privacy-preserving distributed data mining. In *SDM*, pages 265–276. SIAM, 2008.

[31] Charles Duhigg. How companies learn your secrets. *The New York Times*, 16:2012, 2012.

[32] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*. Springer, 2006.

[33] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9 (3–4):211–407, 2014.

[34] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[35] epic. Video privacy protection act (vppa). URL `https://www.epic.org/privacy/vppa/`.

[36] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb): 625–660, 2010.

[37] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[38] Arik Friedman, Bart P Knijnenburg, Kris Vanhecke, Luc Martens, and Shlomo Berkovsky. Privacy aspects of recommender systems. In *Recommender Systems Handbook*, pages 649–688. Springer, 2015.

[39] Johannes Gehrke, Michael Hay, Edward Lui, and Rafael Pass. Crowd-blending privacy. In *Advances in Cryptology–CRYPTO 2012*, pages 479–496. Springer, 2012.

[40] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford, CA, USA, 2009. AAI3382729.

[41] Craig Gentry, Shai Halevi, and Nigel P Smart. Homomorphic evaluation of the aes circuit. In *Advances in Cryptology–CRYPTO 2012*, pages 850–867. Springer, 2012.

[42] Craig Gentry et al. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

[43] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016.

[44] S. Goldwasser and S. Micali. Probabilistic encryption &amp; how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 365–377. ACM, 1982.

[45] Grouplens. Movielens 1m dataset. URL https://grouplens.org/datasets/movielens/1m/.

[46] Rachid Guerraoui, Anne-Marie Kermarrec, Rhicheek Patra, and Mahsa Taziki. D2p: distance-based differential privacy in recommenders. *Proceedings of the VLDB Endowment*, 8(8):862–873, 2015.

[47] Asela Gunawardana and Guy Shani. A survey of accuracy evaluation metrics of recommendation tasks. *Journal of Machine Learning Research*, 10 (Dec):2935–2962, 2009.

[48] Shai Halevi and Victor Shoup. Algorithms in helib. In *International Cryptology Conference*, pages 554–571. Springer, 2014.

[49] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[50] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 173–182. International World Wide Web Conferences Steering Committee, 2017.

[51] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[52] Jingyu Hua, Chang Xia, and Sheng Zhong. Differentially private matrix factorization. In *IJCAI*, pages 1763–1770, 2015.

[53] Xunpeng Huang, Le Wu, Enhong Chen, Hengshu Zhu, Qi Liu, Yijun Wang, and Baidu Talent Intelligence Center. Incremental matrix factorization: A linear feature transformation perspective.

[54] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. Ryoan: A distributed sandbox for untrusted computation on secret data. In *OSDI*, pages 533–549, 2016.

[55] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint arXiv:1803.05961*, 2018.

[56] Nick Hynes, Raymond Cheng, and Dawn Song. Efficient deep learning on multi-source private data. *arXiv preprint arXiv:1807.06689*, 2018.

[57] Arjan Jeckmans, Andreas Peter, and Pieter Hartel. Efficient privacy-enhanced familiarity-based recommender system. In *European Symposium on Research in Computer Security*, pages 400–417. Springer, 2013.

[58] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

[59] Daniel Kifer and Ashwin Machanavajjhala. No free lunch in data privacy. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 193–204. ACM, 2011.

[60] Donghyun Kim, Chanyoung Park, Jinoh Oh, Sungyoung Lee, and Hwanjo Yu. Convolutional matrix factorization for document context-aware recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 233–240. ACM, 2016.

[61] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[62] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *International Colloquium on Automata, Languages, and Programming*, pages 486–498. Springer, 2008.

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

[64] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.

[65] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[66] Jaewoo Lee and Chris Clifton. Differential identifiability. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1041–1049. ACM, 2012.

[67] D. Lemire and A. Maclachlan. Slope one predictors for online rating-based collaborative filtering. In H. Kargupta, J. Srivastava, C. Kamath, and A. Goodman, editors, *Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005*, pages 471–475. SIAM, 2005.

[68] Bin Li, Qiang Yang, and Xiangyang Xue. Transfer learning for collaborative filtering via a rating-matrix generative model. In *Proceedings of the 26th annual international conference on machine learning*, pages 617–624. ACM, 2009.

[69] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670. ACM, 2014.

[70] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *Data Engineering, 2007.*

ICDE 2007. IEEE 23rd International Conference on, pages 106–115. IEEE, 2007.

[71] Yehida Lindell. Secure multiparty computation for privacy preserving data mining. In *Encyclopedia of Data Warehousing and Mining*, pages 1005–1009. IGI Global, 2005.

[72] Jian Liu, Mika Juuti, Yao Lu, and N Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–631. ACM, 2017.

[73] Ziqi Liu, Yu-Xiang Wang, and Alexander Smola. Fast differentially private matrix factorization. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 171–178. ACM, 2015.

[74] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *null*, page 24. IEEE, 2006.

[75] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative instructions and software model for isolated execution. *HASP@ ISCA*, 10, 2013.

[76] Frank McSherry and Ilya 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.

[77] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *Foundations of Computer Science, 2007. FOCS'07. 48th Annual IEEE Symposium on*, pages 94–103. IEEE, 2007.

[78] Andriy Mnih and Ruslan R Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2008.

[79] Bamshad Mobasher, Robin Burke, Runa Bhaumik, and Chad Williams. Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Transactions on Internet Technology (TOIT)*, 7 (4):23, 2007.

[80] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. *IACR Cryptology ePrint Archive*, 2017:396, 2017.

[81] MovieLens. MovieLens Datasets. http://grouplens.org/datasets/movielens/.

[82] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy*. IEEE, 2008.

[83] Kartik Nayak, Xiao Shaun Wang, Stratis Ioannidis, Udi Weinsberg, Nina Taft, and Elaine Shi. Graphsc: Parallel secure computation made easy. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 377–394. IEEE, 2015.

[84] Netflix. Netflix prize dataset. URL https://www.kaggle.com/netflix-inc/netflix-prize-data.

[85] Andrew Ng. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[86] Valeria Nikolaenko, Stratis Ioannidis, Udi Weinsberg, Marc Joye, Nina Taft, and Dan Boneh. Privacy-preserving matrix factorization. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 801–812. ACM, 2013.

[87] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 75–84. ACM, 2007.

[88] Steven J Nowlan and Geoffrey E Hinton. Simplifying neural networks by soft weight-sharing. *Neural computation*, 4(4):473–493, 1992.

[89] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious multi-party machine learning on trusted processors. In *USENIX Security Symposium*, pages 619–636, 2016.

[90] Yuanxin Ouyang, Wenqi Liu, Wenge Rong, and Zhang Xiong. Autoencoder-based collaborative filtering. In *International Conference on Neural Information Processing*, pages 284–291. Springer, 2014.

[91] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptology—EUROCRYPT'99*, pages 223–238. Springer, 1999.

[92] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[93] Weike Pan, Evan Wei Xiang, Nathan Nan Liu, and Qiang Yang. Transfer learning in collaborative filtering for sparsity reduction. In *AAAI*, volume 10, pages 230–235, 2010.

[94] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*, pages 129–140. Springer, 1991.

[95] NhatHai Phan, Yue Wang, Xintao Wu, and Dejing Dou. Differential privacy preservation for deep auto-encoders: an application of human behavior prediction. In *AAAI*, volume 16, pages 1309–1316, 2016.

[96] Huseyin Polat and Wenliang Du. Privacy-preserving collaborative filtering using randomized perturbation techniques. 2003.

[97] Huseyin Polat and Wenliang Du. Privacy-preserving top-n recommendation on horizontally partitioned data. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 725–731. IEEE Computer Society, 2005.

[98] Huseyin Polat and Wenliang Du. Achieving private recommendations using randomized response techniques. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 637–646. Springer, 2006.

[99] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

[100] Michael O Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.

[101] recombee. Recommender as a service. URL https://www.recombee.com/.

[102] PJ Rossky, JD Doll, and HL Friedman. Brownian dynamics as smart monte carlo simulation. *The Journal of Chemical Physics*, 69(10):4628–4633, 1978.

[103] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: Scalable provably-secure deep learning. *arXiv preprint arXiv:1705.08963*, 2017.

[104] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798. ACM, 2007.

[105] Issei Sato and Hiroshi Nakagawa. Approximation analysis of stochastic gradient langevin dynamics by using fokker-planck equation and ito process. In *ICML*, 2014.

[106] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web*, pages 111–112. ACM, 2015.

[107] Erez Shmueli and Tamir Tassa. Secure multi-party protocols for item-based collaborative filtering. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 89–97. ACM, 2017.

[108] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321. ACM, 2015.

[109] Reza Shokri, Pedram Pedarsani, George Theodorakopoulos, and Jean-Pierre Hubaux. Preserving privacy in collaborative filtering through distributed aggregation of offline profiles. In *Proceedings of the third ACM conference on Recommender systems*, pages 157–164. ACM, 2009.

[110] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 3–18. IEEE, 2017.

[111] Qiang Song, Jian Cheng, and Hanqing Lu. Incremental matrix factorization via feature space re-learning for recommender system. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 277–280. ACM, 2015.

[112] Shuang Song, Kamalika Chaudhuri, and Anand D Sarwate. Stochastic gradient descent with differentially private updates. In *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, pages 245–248. IEEE, 2013.

[113] Florian Strub and Jérémie Mary. Collaborative filtering with stacked denoising autoencoders and sparse inputs. In *NIPS workshop on machine learning for eCommerce*, 2015.

[114] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4, 2009.

[115] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10 (05):557–570, 2002.

[116] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[117] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[118] Q. Tang. Cryptographic framework for analyzing the privacy of recommender algorithms. In *2012 International Symposium on Security in Collaboration Technologies and Systems (CTS 2012)*, pages 455–462, 2012.

[119] T. Veugen. Comparing encrypted data. http://bioinformatics.tudelft.nl/sites/default/files/Comparing2011.

[120] João Vinagre, Alípio Mário Jorge, and João Gama. Fast incremental matrix factorization for recommendation with positive-only feedback. In *International Conference on User Modeling, Adaptation, and Personalization*, pages 459–470. Springer, 2014.

[121] Sebastian J Vollmer, Konstantinos C Zygalakis, et al. (non-) asymptotic properties of stochastic gradient langevin dynamics. *arXiv preprint arXiv:1501.00438*, 2015.

[122] H. Wang. https://github.com/luxjwang/Experiments/tree/master/code2016/tdsc/cryptonbm.

[123] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1235–1244. ACM, 2015.

[124] Yu-Xiang Wang, Stephen E Fienberg, and Alex Smola. Privacy for free: Posterior sampling and stochastic gradient monte carlo. *Blei, D., and Bach, F., eds*, 951(15), 2015.

[125] Udi Weinsberg, Smriti Bhagat, Stratis Ioannidis, and Nina Taft. Blurme: inferring and obfuscating user gender based on ratings. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 195–202. ACM, 2012.

[126] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 681–688, 2011.

[127] wiki. Robert bork supreme court nomination. URL https://en.wikipedia.org/wiki/Robert_Bork_Supreme_Court_nomination.

[128] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057, 2015.

[129] I. Yakut and H. Polat. Arbitrarily distributed data-based recommendations with privacy. *Data & Knowledge Engineering*, 72(0):239 – 256, 2012.

[130] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

[131] Jun Zhang, Zhenjie Zhang, Xiaokui Xiao, Yin Yang, and Marianne Winslett. Functional mechanism: regression analysis under differential privacy. *Proceedings of the VLDB Endowment*, 5(11):1364–1375, 2012.

[132] Shuai Zhang, Lina Yao, and Aixin Sun. Deep learning based recommender system: A survey and new perspectives. *arXiv preprint arXiv:1707.07435*, 2017.

[133] Shuai Zhang, Lina Yao, and Xiwei Xu. Autosvd++: An efficient hybrid collaborative filtering model via contractive auto-encoders. *arXiv preprint arXiv:1704.00551*, 2017.

[134] Tianqing Zhu, Gang Li, Yongli Ren, Wanlei Zhou, and Ping 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.

[135] Tianqing Zhu, Yongli Ren, Wanlei Zhou, Jia Rong, and Ping Xiong. An effective privacy preserving algorithm for neighborhood-based collaborative filtering. *Future Generation Computer Systems*, 36:142–155, 2014.