UNIVERSITY OF CALIFORNIA

Los Angeles

Emerging Paradigms for Privacy Preserving Recommender Systems

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Manoj Reddy Dareddy

2022

ABSTRACT OF THE DISSERTATION

Emerging Paradigms for Privacy Preserving Recommender Systems

by

Manoj Reddy Dareddy
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2022
Professor Junghoo Cho, Chair

Recommender Systems are ubiquitous on the web. They are used to recommend users with movies to watch, songs to listen to, products to buy etc. The main goal of recommender systems is to personalize a user's experience based on their interests conveyed through historical feedback information. Existing recommender systems are centralized in nature, that requires a server to collect detailed item feedback information across the entire user population. This status quo presents serious privacy concerns since the central recommendation server has access to fine-grained preference information about each individual user. This feedback information can be utilized to infer the user's sensitive information such as their medical condition, religious, political affiliation etc. This leads to serious privacy concerns. Moreover, a small fraction of users who are aware of such privacy risks tend to share less feedback which in turn reduces the quality of their recommendations.

Since privacy is being recognized as a fundamental human right, it's imperative that personalization systems provide recommendations in a privacy-preserving manner. In this thesis, we present three privacy-preserving recommender system paradigms whereby the amount the information sent to the central recommender system is gradually reduced.

Paradigm I provides privacy preserving session-based item recommendations where the central server simulates an incognito behavior by treating each user as anonymous. The framework relies on item sequence information across sessions to recommend the next item for a user. Paradigm II enables the user to decide which ratings they would like to keep private i.e., store locally on-device vs public i.e., share with the central recommendation server. Public ratings from all users are used to build a global model and then fine-tuned on each user's device based on their private ratings. Paradigm III enables each user to store their entire feedback information on-device and employs a federated learning mechanism to perform private learning for recommendation. Differential privacy is used to quantify the privacy budget for an individual user. Each of the three paradigms are scalable to the industrial setting and more importantly, empower each user to determine their individual privacy policy for their recommendations.

To my parents: Laxma Reddy Dareddy & Sarala Devi

Table of Contents

1 Introduction **1**

2 Privacy in Recommender Systems **4**

2.1 Status Quo 4

2.2 Importance of Privacy 5

2.3 Research Question 5

3 Bayesian Prior Learning via Neural Networks for Next-item Recommendation . . . **6**

3.1 Introduction 7

3.2 Related Work 9

3.3 Our Approach 10

3.3.1 Intuition 10

3.4 Experimental Setup 17

3.4.1 Dataset 17

3.4.2 Training Process 20

3.4.3 Candidate Generation 22

3.4.4 Evaluation Metrics 22

3.4.5 Baselines 23

3.5 Results 26

3.6 Discussion 26

3.7 Conclusion 27

4 Selective Privacy Preserving Collaborative Filtering 29

4.1 Introduction 30

4.1.1 Motivation 31

4.1.2 Contributions 32

4.2 User Survey: Privacy Concerns 32

4.3 SP2 Architecture 34

4.3.1 Data Sharing Model 34

4.3.2 Collaborative Filtering (CF) Algorithm 35

4.3.3 Problem Formulation 37

4.3.4 Architecture 38

4.4 Public Item Factor Matrix and CF Algorithm 40

4.4.1 Naive Approach 40

4.4.2 Clustering 42

4.4.3 Joint Optimization 45

4.5 Experiments 47

4.5.1 SP2 vs. Different Baselines 47

4.5.2 Private Ratings Allocation 48

4.5.3 Evaluation Setup 49

4.5.4 Results 51

4.6 Related Work 56

4.7 Conclusion 57

5 Federated Local Differential Privacy for Recommender Systems 58

5.1 Introduction 58

5.2 Background 60

5.2.1 Collaborative Filtering 60

5.2.2 Federated Learning 61

5.2.3 Local Differential Privacy 62

5.3 Motivation 62

5.4 Threat Model 63

5.4.1 Role of Central Server 63

5.5 Practical Implications of Differential Privacy 63

5.6 TensorFlow Privacy 65

5.6.1 Individual-level user privacy 66

5.6.2 Appropriate ε value 66

5.7 Proofs 67

5.7.1 Proof I: Sequential Composition of Users across Datasets 67

5.8 Framework 69

5.8.1 Individual Privacy Controls 69

5.8.2 Stochastic Gradient Descent 69

5.8.3 Client Side Computation 69

5.8.4 Server side Computation 71

5.9 Results 72

5.10 Related Work 73

5.10.1 Privacy Preserving Recommendation 73

5.10.2 Differential Privacy 76

6 Conclusion and Future Directions 78

6.1 Computational and Communication Efficiency 79

6.2 Robustness 79

6.3 Group Privacy 79

References 81

List of Figures

3.1 Status Quo of Next Item Recommendation 7

3.2 Longer context leads to better prediction accuracy 12

3.3 Higher frequency indicates higher confidence 13

3.4 Modeling the posterior distribution instead of raw point estimate 14

3.5 Proposed Neural Network Architecture 17

3.6 Detailed Alpha/Beta Function 18

3.7 Playlist Length Frequency Distribution 19

3.8 Track frequency distribution across playlists 20

4.1 Selective privacy preserving (SP2) framework from a user's perspective. 31

4.2 How often do you hesitate to rate an item or write a review because you do not want to share your opinion publicly or trust the platform? 33

4.3 Would you rate more items if you can turn off public sharing and store your rating *privately* only on your device to improve the quality of recommendations? 33

4.4 Architecture of a SP2 framework. 39

4.5 Probability density functions of four different beta distributions used in private ratings allocation. 50

4.6 Results of SP2 52

4.7 RMSE comparison 53

4.8 NDCG comparison 53

4.9 Aux. data size comparison 54

4.10 RMSE comparison 54

5.1 Motivation of Differential Privacy 64

5.2 Stochasticity of a differentially private algorithm 64

5.3 Our Proposed Framework 68

5.4 Individual Privacy Controls 68

5.5 Epsilon vs RMSE on Movielens 72

5.6 Test RMSE over epochs in Movielens 73

5.7 Epsilon vs RMSE on Amazon Video 74

5.8 Test RMSE over epochs in Amazon 74

List of Tables

3.1 Scenario 1 Frequency Values 10

3.2 Scenario 2 Frequency Values 11

3.3 Results for the Spotify Dataset 26

3.4 Results for the Movielens dataset 26

4.1 Definitions of symbols used in (5.1) - (5.6) 36

ACKNOWLEDGMENTS

Let me begin by first thanking my advisor, Professor John Cho, for his guidance throughout my doctoral studies. I am immensely grateful to him for giving me an opportunity to pursue a PhD and also offering me great flexibility to work on different research topics. He has been a tremendous role model and has always provided me with excellent advice and support throughout my studies. I am inspired by Prof. Cho's approach to solving problems and amazed by his intuition and ability to deconstruct complex ideas using simple terminology.

I would also like to express my deep gratitude to my committee, Professor Wei Wang, Professor Yingnian Wu and Professor Carlo Zaniolo, for their time, help and wonderful suggestions over the years. During my time at UCLA and my summers at Samsung Research America, Visa Research and Google, I also had the opportunity to work closely with many other amazing people. In particular, I would like to thank Rui Chen, Mahashweta Das and Vartan Akopian for mentoring me during my internships and making this journey even more enjoyable and rewarding.

I am also grateful to all my friends and fellow students at UCLA for helping me along the way. This list includes but is not limited to Ariyam Das, Himel Dev, Mingda Li, Youfu Li, Qiujing Lu, Tonmoy Monsoor, Uneeb Rathore, Arash Vahabpour, Tianyi Wang, Jin Wang, Zijun Xue . I am also very thankful to our department staffs, especially Juliana Alvarez, Steve Arbuckle and Joseph Brown, who always promptly responded whenever I needed help.

I am infinitely grateful to my parents, Laxma Reddy Dareddy and Sarala Devi Gottamu, for their unconditional love and unwavering support. Without their constant encouragement, I would not have been able to embark on this wonderful journey. They are my greatest teachers, and whatever I may have accomplished is a credit to them. Thank you, Mom and Dad.

Last but not least, I am very grateful for the support from Qatar Foundation. I would like to especially thank Dr. Aisha Al-Obaidly, Dr. Ayman Bassil, Maria Susi Estacio and the entire team at Qatar Foundation Research & Development Division.

VITA

2013	Bachelor of Computer Science, Carnegie Mellon University in Qatar.
2014	Software Engineer Intern, Amazon, Seattle, WA.
2015	Master of Science in Information, University of Michigan, Ann Arbor.
2015-2022	Graduate Student Researcher, Department of Computer Science, UCLA.
2016-2017	Teaching Assistant, Department of Computer Science, UCLA.
2017	Data Scientist Intern, Samsung Research America, Mountain View, CA.
2017-2018	Teaching Associate, Department of Computer Science, UCLA.
2018	Research Scientist Intern, Visa Research, Palo Alto, CA.
2019	Software Engineer Intern, Google, Los Angeles, CA.
2018-2022	Teaching Fellow, Department of Computer Science, UCLA.

PUBLICATIONS

• Lacic, E., Reiter-Haas, M., Kowald, D., **Dareddy, M. R.**, Cho, J., & Lex, E. (2020). Using autoencoders for session-based job recommendations. User Modeling and User-Adapted Interaction, 30(4), 617-658.

• **Dareddy, M. R.**, Das, M., & Yang, H. (2019, December). motif2vec: Motif aware node representation learning for heterogeneous networks. In 2019 IEEE International Conference on Big Data (Big Data) (pp. 1052-1059). IEEE.

• Vydiswaran, V. V., & **Reddy, M.** (2019). Identifying peer experts in online health forums. BMC medical informatics and decision making, 19(3), 41-49.

• **Reddy, M.**, & Vydiswaran, V. V. (2018, June). Towards Identifying Peer Expertise in Online Health Forums. In 2018 IEEE International Conference on Healthcare Informatics Workshop (ICHI-W) (pp. 82-83). IEEE.

• **Reddy, M.** (2018, March). Tackling item coldstart in recommender systems using word embedding. In Qatar Foundation Annual Research Conference Proceedings Volume 2018 Issue 3 (Vol. 2018, No. 3, p. ICTPD346). Hamad bin Khalifa University Press (HBKU Press).

• **Dareddy, M. R.** (2017, February). Recommender Systems: Research Direction. In Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (pp. 831-831).

• Mishra, S. K., & **Reddy, M.** (2016). A bottom-up approach to job recommendation system. In Proceedings of the Recommender Systems Challenge (pp. 1-4).

• **Reddy, M.**, & Cho, J. (2016, March). Detecting chronic kidney disease using machine learning. In Qatar Foundation Annual Research Conference Proceedings Volume 2016 Issue 1 (Vol. 2016, No. 1, p. ICTSP1534). Hamad bin Khalifa University Press (HBKU Press).

• **Dareddy, M. R.** (2016). Challenges in Recommender Systems for Tourism. In RecTour@ RecSys (pp. 59-61).

CHAPTER 1

Introduction

Recommender Systems drive the user engagement of various platforms on the web. They are employed in domains such as: movies, songs, retail products to assist the user to decide the item to explore. In addition to being explicitly employed for personalization to a user, recommender systems are used implicitly for various critical components of a website such as ranking, selection and filtration of items. An example of such a system is Quora, a question-answering based forum whereby users can ask questions and receive answers from the community. Recommender Systems are utilized for various aspects of personalization such as: deciding which questions to display on the homepage, the ranking of answers for a given question, friend recommendation, to name a few. Recommender Systems are a core component of platforms on the web since they reduce churn rate and help users discover personalized content.

Current recommender systems rely on the collaborative filtering approach which requires users to share their information to the central recommendation server. This server collects preference information from millions of users and utilizes correlation across users to generate personalized recommendations. This poses a privacy concern since the server has detailed information about the preferences of a user. It can utilize this information to accurately predict the health status of a user and stance on sensitive topics such as religion, political affiliation etc. Moreover, the data generated by a user belongs to the user and hence he/she must have control over it is being used. Hence, there is a need for recommender systems to be less intrusive i.e. servers that don't necessarily collect all preference information from users and provide privacy-preserving recommendations.

In this thesis, we develop three paradigms/frameworks that provide privacy preserving recommendations in an incremental fashion. The initial framework, ‘Bayesian Prior Learning via Neural Networks for Next-Item Recommendation’ aims to provide anonymous next-item recommendations. The sequence of items interacted in a session is useful information for recommending the next item. In order to mine this sequence information in a privacy preserving manner, we disassociate the sequence information from the users, i.e. *de-anonymize* the sequences from users and compute the frequencies of various subsequences. The subsequence frequency information can be used to predict the likelihood of a sequence using the Beta distribution. We demonstrate the effectiveness of our approach with several baselines including some non-personalized versions of existing state-of-the-art approaches. This framework simulates the scenario whereby user data is stored on the central server but the central recommender system guarantees to remove any correlation between user identifier and the items consumed in the session. This framework requires users to trust the central recommender server to provide an "incognito" behavior.

In the first framework, we demonstrate a privacy-preserving recommendation framework that requires users to trust the central server their entire feedback data. In our next framework, ‘Selective Privacy Preserving Collaborative Filtering’, we aim to give more control to the user to decide which portion of their data they would like to keep as private. By keeping a portion of data private, the users are ensured that it stays on their device and is never shared with anyone else. Our framework builds a global model using the public information gathered across all users. And each user then refines their individual model using the private ratings stored on their device. The final model is then never shared with the server and stored locally on the user’s device. In our experiments, we simulated various scenarios of user behavior in their privacy levels and demonstrated that our approach performs much better than existing baselines in the evaluation metrics on two real-world datasets. Compared to the first framework, our second framework empowers users by allowing them to determine their individual privacy policy.

In our final framework, we would like to store the entire user data on their devices locally. For this purpose, we employ the notion of Differential Privacy to quantify the privacy budget of each user. Differential Privacy provides rigorous mathematical guarantees on the quantification of privacy loss by an algorithm. The essential idea is that just by observing the output, it's difficult for an external party to infer the input of the algorithm. We apply the case of local differential privacy to the federated case of collaborative filtering. Our framework employs the federated learning approach, whereby in each round the client updates the gradients of the corresponding user and item latent factors. We experimentally demonstrate the effectiveness of our approach on 2 real world datasets and illustrate the trade-offs with regards to privacy vs. accuracy. The three frameworks presented incrementally provide higher privacy controls to the user since in Framework I, users are required to trust that the central recommendation server provides privacy guarantees by de-linking the session information and individual user identity. In Framework II, users are able to individually decide their privacy policy by determining which ratings they would like to store as public/private. In Framework III, entire user data remains private by storing it locally on their device. Each user decides their individual differential privacy budget that determines the amount of noise to added for the gradient updates in federated learning setting.

The rest of this thesis is organized as follows:

In chapter 3, we outline the next-item recommendation problem and present our privacy-preserving approach using Bayesian modeling.

Then in chapter 4 we present the second framework of selective privacy preserving whereby the user has more control over which portion of their data is shared with the central recommendation server.

In chapter 5, we present our final framework that allows users to store their entire data locally on their device. We employ differential privacy to ensure each user can set their individual privacy budget and the learning occurs in a federated fashion whereby user only shares noise-added gradients to the server.

Finally, we conclude and discuss future research directions originating from this work in Chapter 6. This thesis aims to motivate the importance and proposes three frameworks for practical, scalable and effective privacy-preserving recommender systems. We hope our paradigms motivate researchers to extend our work on privacy preserving recommender systems.

CHAPTER 2

Privacy in Recommender Systems

2.1 Status Quo

Existing recommender systems are centralized in nature. Typically, there exists a Central Recommender Server (CRS) that is responsible for providing recommendations to a large number of users. The CRS is responsible for selecting a small subset from a large set of items, that would be most relevant to a particular user. In order to receive accurate recommendations, users have to share their preference information via implicit or explicit feedback to the CRS. Under the current setup, the entire feedback information about the likes and dislikes of a user resides on the CRS. This fine-grained user feedback can enable the CRS to accurately predict the user preferences on sensitive topics such as their: health, religion, political affiliation, gender identity, substance abuse etc. This results in serious privacy concerns since the user is vulnerable to the information collected by the CRS. Majority of the users of recommender systems are unaware of such privacy concerns. But for the users who are aware, such privacy concerns prevent them from providing feedback to the CRS. This leads to a decrease in the quality of recommendations since there is an inherent trade-off between accuracy vs privacy in recommender systems. The more feedback a user shares with the CRS, the better his/her recommendations will result in lower privacy since the CRS has access to more information about that user. Similarly, when a user shares less with the CRS, their quality of recommendations drop but they have better privacy since they share less information with the CRS.

2.2 Importance of Privacy

Generally, privacy is described in the context of security. Although, the two concepts are inter-related, there exists a fine line of distinction between privacy and security. Computer security deals with protecting data from unauthorized access whereas privacy is about using user data responsibly. Most of the CRS in industry have very good security policies in place but the same cannot be said about their privacy policies. In recent times, government regulators have levied large fines against firms that have violated privacy laws such as the California Consumer Privacy Act (CCPA) and General Data Protection Regulation (GDPR) based in Europe. As the awareness about privacy grows amongst the public, it is in the best interest of the CRS to provide privacy-preserving recommendations to it's users. In this thesis, we propose scalable frameworks that enable a win-win situation for both the CRS and the millions of users by providing privacy-preserving recommendations.

2.3 Research Question

In this thesis, the central research question we aim to answer is that can we reduce the amount of data shared to the Central Recommender Server (CRS), meanwhile generate high-quality recommendations? As motivated earlier, this is a crucial question to answer since existing CRS are data hungry and infringe upon user privacy. We propose 3 frameworks that enable the CRS to provide privacy preserving recommendations. Our proposed frameworks are incremental in nature, i.e. they gradually share less feedback information from users to the CRS. Moreover, our frameworks empower the user to decide their individual privacy policy. Our research work can broadly be classified under the direction of federated learning where the challenge is to perform a learning task when the data is stored in a distributed fashion.

CHAPTER 3

Bayesian Prior Learning via Neural Networks for Next-item Recommendation

Next-item prediction is a a popular problem in the recommender systems domain. As the name suggests, the task is to recommend subsequent items that an user would be interested in given contextual information and historical interaction data. In our paper, we model a general notion of context via a sequence of item interactions. We model the next item prediction problem using the Bayesian framework and capture the probability of appearance of a sequence through the posterior mean of the Beta distribution. We train two neural networks to accurately predict the alpha beta parameter values of the Beta distribution. Our novel approach of combining black-box style neural networks, known to be suitable for function approximation with Bayesian estimation methods have resulted in an innovative method that outperforms various state-of-the-art baselines. We demonstrate the effectiveness of our method in the song recommendation domain using the Spotify playlist continuation dataset.

Existing next-item prediction approaches are personalized, i.e. they model user behavior based on historical interactions with the item set. State-of-the-art neural-network based methods perform very well since they model individual users/items via embeddings and capture their sequential consumption patterns fairly accurately.

Our approach on the other hand is a novel privacy-preserving approach to the next-item prediction problem. We do not model individual users per se, since we are only gather statistical

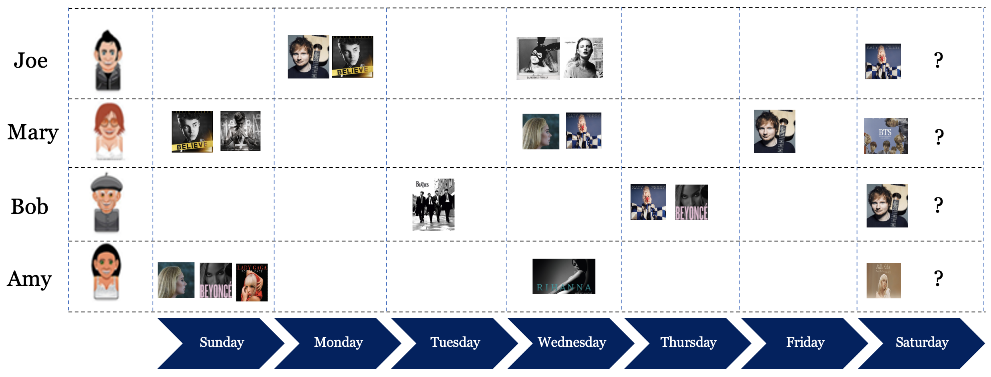


Figure 3.1: Status Quo of Next Item Recommendation

information about n-grams.

Current recommender systems used in industry perform very well in being able to capture an user's interest and predict future items they would be interested in. But this occurs at the expense of user privacy since vast amounts of information are being logged by these systems in order to accurately model the interests of an user. These data-hungry models aim to model users via rich-contextual information hence infringing user privacy in general.

Such invasive data collection practices discourages users from participating on the platform. Instead, our framework encourages users to freely participate since we do not model any specific individual user or item.

3.1 Introduction

Next-item prediction is a classic problem in the domain of Natural Language Processing (NLP) and Recommender Systems. In the NLP domain, it is used for the purpose of language modeling [JVS16] whereby the task is given a set of tokens, predict the most likely next token. In natural language, words can be represented by tokens and accurately predicting the next token helps us understand the intricacies of a particular language. In Recommender Systems, we maybe given access to historical transactions by a set of users such as products purchased, movies watched etc. The next item prediction problem in this domain [AT05] boils down to the task of predicting the next item/movie a user is most likely to purchase/watch.

As illustrated in Figure 3.1, existing next-item recommender systems track user behavior over time, and utilize this information to provide recommendations of items they are most likely to consume in a session. Recommender Systems that track the entire user history in general are able to provide high-quality recommendations since they do so at scale across millions of users. The drawback of such systems, is that the accuracy comes at the expense of user privacy, since each individual user's activity is being logged by the system. Moreover, users do not have control over how their data is being utilized on the server which results in the loss of their privacy.

Various approaches have been proposed in the literature for the next-item prediction problem. Majority of these techniques rely on an unsupervised training dataset which captures previous interactions. For instance, in the NLP domain it can refer to a large corpus of unstructured documents, whereby sentences are represented as a sequence of tokens/words. In the recommender systems domain it can refer to sequences of historical purchases of items that are organized in sessions.

Majority of the existing approaches tackle the next item problem using statistical techniques by analyzing for patterns across the provided training dataset. By sifting through large amounts of training data, these models are able to fairly accurately predict the next item at test time given a seed context.

The proposed approaches fall under the following two categories: black-box neural network and hand-crafted approaches. The former set of approaches employ the use of neural networks which are known to have high representational power and perform implicit feature engineering. Among the latter miscellaneous hand-crafted approaches, Bayesian methods are known to be intuitive and easy to understand/debug. Each approach has their own set of pros and cons. Neural network approaches perform very well in practice but have the drawback of being considered as a black-box, i.e. provides limited understanding of the underlying prediction mechanism. Bayesian approaches on the other hand are based on the Bayes rule [Sto13], a fundamental concept in probability making it intuitive and easy to understand. The drawback of a pure Bayesian approach is that it does not perform as well as the neural network approaches in the task of next-item prediction.

In this work, our main contribution is as follows:

- We propose a Bayesian framework with prior information learnt using neural networks for the next-item prediction.
- We introduce the confidence of observed data using the Beta distribution and utilize a Siamese network to estimate the Beta model parameters.
- We demonstrated the performance improvements our method against existing state-of-the-art approaches.

The rest of the paper is organized as follows. In section 2, we describe the related work existing in the literature. In section 3, we describe our approach in detail along with our motivation and intuitions. In section 4, we describe the experimental setup and detailed training process. In section 5, we present our results in comparison with various state-of-the-art baselines. In section 6, we analyze our results and provide some insights into our understanding of the problem. In section 7, we propose further directions that can be pursued to expand this research area. In section 8, we conclude by highlighting our novel contribution to the next-item prediction problem.

3.2 Related Work

The related work in this area consists of two main categories, namely: neighborhood based and neural network based approaches. There exist other types of approaches which we describe later in this section.

Neural network based approaches mostly reply on the sequence based models using the Recurrent Neural Network architecture. Li et al. [LZL18] propose an item embedding method based on an aggregation of the user interactions. They then use a contextual LSTM neural network architecture to train on two real-world datasets in order to predict the next-item. Wu et al. [WHS19] expand on attention mechanism idea proposed by Vaswani et al. [VSP17] which is known to be efficient and also work well in the natural language processing domain. They were able to demonstrate the superiority of the attention mechanism over the LSTM approach.

In addition to above mentioned approaches, there exist unconventional yet effective methods to recommendation under different settings. For example, Dareddy et al. [DDY19] propose the use of motifs in heterogeneous networks for the task of link recommendation. Similarly, Mishra et al. [MR16] proposed a novel next-item prediction solution tailored towards job recommendation whereby users were matched with jobs based on a variety of context features.

3.3 Our Approach

3.3.1 Intuition

Our problem can be formulated as follows: $\mathscr{I} = \{a, b, c...\}$ represents a set of items, $\mathscr{S} = (S_1, S_2, ...S_n)$ represents a user interaction sequence with $S_i \in \mathscr{I}$. We observe many user sequences and the goal is given a length k subsequence C of S, $C = (S_i, S_{i+1}, ...S_{i+k-1}) \subset S$, predict the next item in C i.e. to accurately estimate the following probability for a specific item a:

$$P(S_{i+k} = a|(S_i, S_{i+1}, ...S_{i+k-1})) \tag{3.1}$$

An accurate estimation of the above probability requires analyzing our observed training data. The intuition is that given a longer context information, perhaps we can more accurately compute the above posterior probability value.

Assume we are given a context of 3 items, namely (a, b, c) and we are interested in calculating the next-item probability values for 2 items namely: x and y. Hence we are interested in computing the following values:

$$P(x|(a, b, c)) \ \& \ P(y|(a, b, c))$$

Consider two scenarios for the frequency of n-gram values in our training dataset. Scenario 1 is described as below:

Prefix	abc		bc		c		-	
Sequence	abcx	abcy	bcx	bcy	cx	cy	x	y
Frequency	100	50	200	100	300	150	500	1000

Table 3.1: Scenario 1 Frequency Values

Based on the frequency values in Table 3.1, we observe that in general item y is more popular than item x since their frequencies in the train dataset are 1000 and 500 respectively. But under various contexts {(c), (b, c) & (a, b, c)}, we see that item x appears more than item y. This indicates that item x would be a better choice than item y for next item prediction given context (a, b, c).

Consider another Scenario 2 whereby the frequency information of sequences is as follows:

Prefix	abc		bc		c		-	
Sequence	abcx	abcy	bcx	bcy	cx	cy	x	y
Frequency	2	1	5	500	50	750	500	1000

Table 3.2: Scenario 2 Frequency Values

Similar to Scenario 1, in general item y is popular than item x (same frequencies as before). In contrast to the previous scenario, for the context (a, b, c), item x seems to be a better choice than item y, due to the frequencies being 2 and 1 respectively. Although item x appears twice as many times as item y under the context (a, b, c) it is not a reliable signal since the raw frequency values are very low. This results in a low confidence of accuracy for likelihood estimation under the context (a, b, c). Instead if we rely on a smaller context (b, c), we observe that item y is clearly a better choice than item x since it co-occurs 500 compared to 5 times respectively. We observe the same phenomenon under the context of single item (c). Overall, the observations under smaller contexts can be considered to be more confident in recommending item y as the better next item recommendation under this scenario.

Consider a scenario as shown in Figure 3.2 whereby we are tracking the items a user has purchased in a given session. Given that the user has just purchased a bagpack, the systems recommends items that are related to the initial purchase, such as: waterbottle, diary, hiking ropes, calculator, umbrella and trekking poles. These items would be considered good recommendations since we are not sure whether the user is buying a bagpack for school, travel or hiking purposes. In the next column, the user purchases a diary after the bagpack purchase, indicating that the user is likely to be interested in purchasing items that are school-related. We can be more confident about the items to recommend once we know that the user has purchased a bagpack followed by diary and a calculator. In the case, we can be confident that the user is interested in purchasing stationary supplies, hence items such as stapler, folders etc. would be good recommendations. The

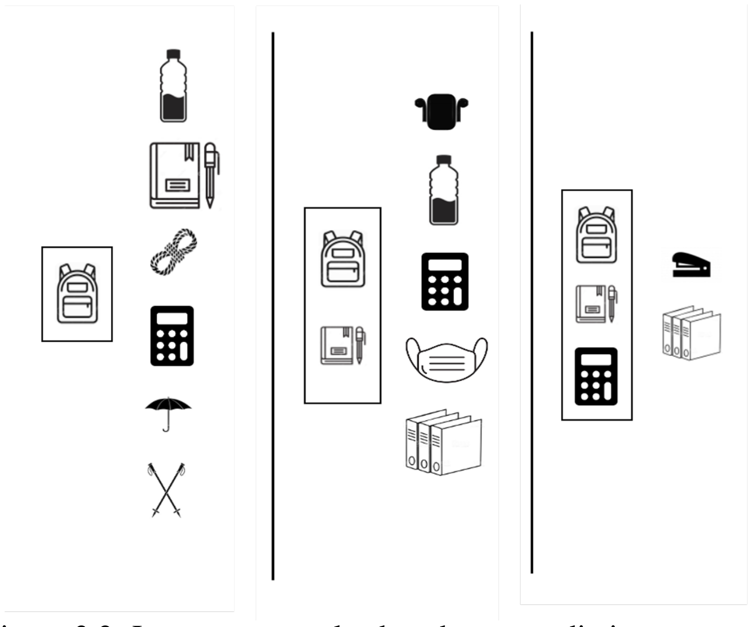


Figure 3.2: Longer context leads to better prediction accuracy

main intuition here is that the more context information we have, the better we can predict the next item, simply because we have access to more information from the context.

Consider another scenario whereby we are analyzing the purchases made by users in a system. Let's say we observe that out of 10 purchase sequences of a laptop followed by a mobile, we observe that users have purchased a watch 9 times. This implies that it's likely that given a sequence of laptop and mobile purchase, the user is likely to buy a watch about 90% of the transactions. In the second scenario, given that we observe only one transaction across the entire transactions log of the following of items: computer, bagpack and a music label. In the future, given that the user has purchased a PC followed by a bagpack, then the next item to recommend based on our history would be the music label with a probability of 100%. In reality, the music label is not a good recommendation since we observe the entire transaction only once. The issue here is that of data sparsity. Given a limited set of observations, it is inherently difficult to estimate the likelihood probability. In order to mitigate the challenge of data sparsity, we capture our belief of the probabilities using the posterior distribution for every item. Intuitively, instead of modeling the point estimates based on ratios obtained from historical transactions, we ought to model the distribution of the point estimate based on the raw frequency counts of sequences observed. In

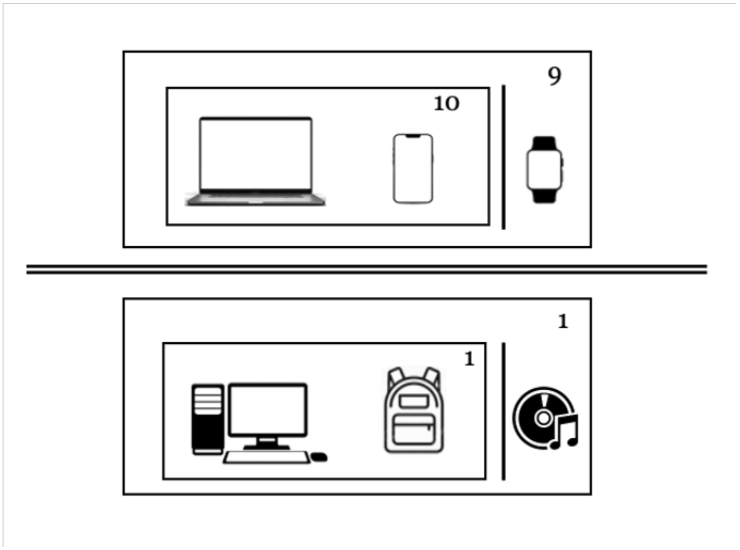


Figure 3.3: Higher frequency indicates higher confidence

particular we use the Beta distribution.

Beta distribution is well known to be a good prior for various distributions such as Binomial, Normal and is widely used in practice as well [ACD11]. We chose the Beta distribution since it is able to effectively model binary feedback. The Beta distribution consists of 2 parameters, namely: alpha (α) & beta (β). The parameter alpha captures the positive count whereas the beta distribution captures the corresponding negative count. The mean of a Beta distribution summarizes the overall binary feedback.

Given a set of α & β values, the mean of the Beta distribution is as follows:

$$\mu(\alpha, \beta) = \frac{\alpha}{\alpha + \beta} \tag{3.2}$$

The mean of the Beta distribution can provide insights into the overall likelihood of observing a given sequence.

Our goal is to incorporate the counts of various context frequencies so that the neural network can learn the most appropriate α & β values from our training data. The mean based on alpha & beta values will estimate our likelihood for the probabilities of various context, item pairs.

Without loss of generality, we restrict ourselves to a context size of four items/songs. Given

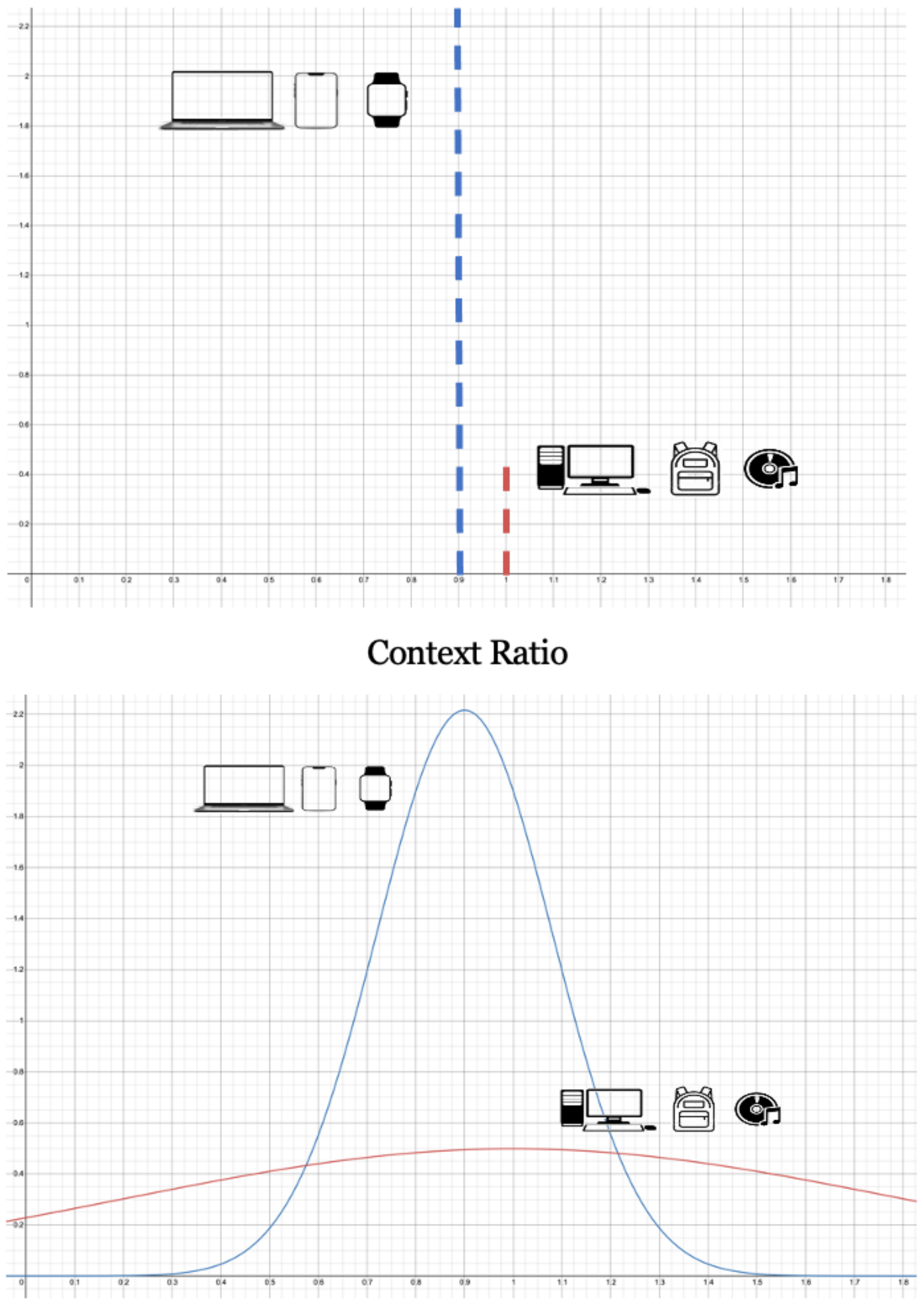


Figure 3.4: Modeling the posterior distribution instead of raw point estimate

a sequence of 4 songs a, b, c, d, we would like to be able to accurately estimate the posterior probability of these 4 songs in order. By doing so, given a test seed playlist of 3 songs and a set of candidates, we can rank the candidates based on the posterior mean and recommend the one with the highest value.

We aim to learn 2 separate functions to accurately predict the alpha and beta values. The idea to ingest frequency counts from the training datasets and learn to accurately predict the alpha and beta values.

Given we observe a sequence of 4 songs ‘a, b, c, d’ in our dataset, we initially collect the following counts/frequency values:

Positive Counts:

- #(d)
- #(cd)
- #(bcd)
- #(abcd)

Negative Counts:

- (∼d)
- #(c∼d)
- #(bc∼d)
- #(abc∼d)

The above values are used as input to the functions that predict the alpha & beta parameter values. We aim to learn these functions in a *contrastive* learning method. For an observed sequence of 4 items in the training data, which we denote as a positive example, we would like to use a negative example so that the functions can learn to maximize the posterior Beta distribution mean between the two.

Our methodology for obtaining a negative example given an observed positive sequence of ‘a, b, c, d’ is as follows:

- If ‘d’ is the most popular song given the prefix ‘a, b, c’ then we select ‘a, b, c, e’ to be the negative example whereby ‘e’ is the next most popular song after ‘d’.
- If ‘d’ is not the most popular song given the prefix ‘a, b, c’ then we select ‘a, b, c, e’ to be the negative sequence whereby ‘e’ is the most popular song given the prefix ‘a, b, c’.

The intuition is that we would like to maximize the difference between the posterior means of the positive and negative samples which are computed using the alpha and beta functions. The alpha and beta functions are learning to predict the accurate alpha and beta parameter values for the positive and negative examples based on their frequency counts as mentioned in the above table.

Without loss of generality, given positive example ‘a, b, c, d’ and the corresponding negative example ‘a, b, c, e’ which is a single training instance to the overall neural network, we initially compute the following four values:

$\alpha_+ = f_\alpha(positive\_example\_counts)$

$\beta_+ = f_\beta(positive\_example\_counts)$

$\alpha_- = f_\alpha(negative\_example\_counts)$

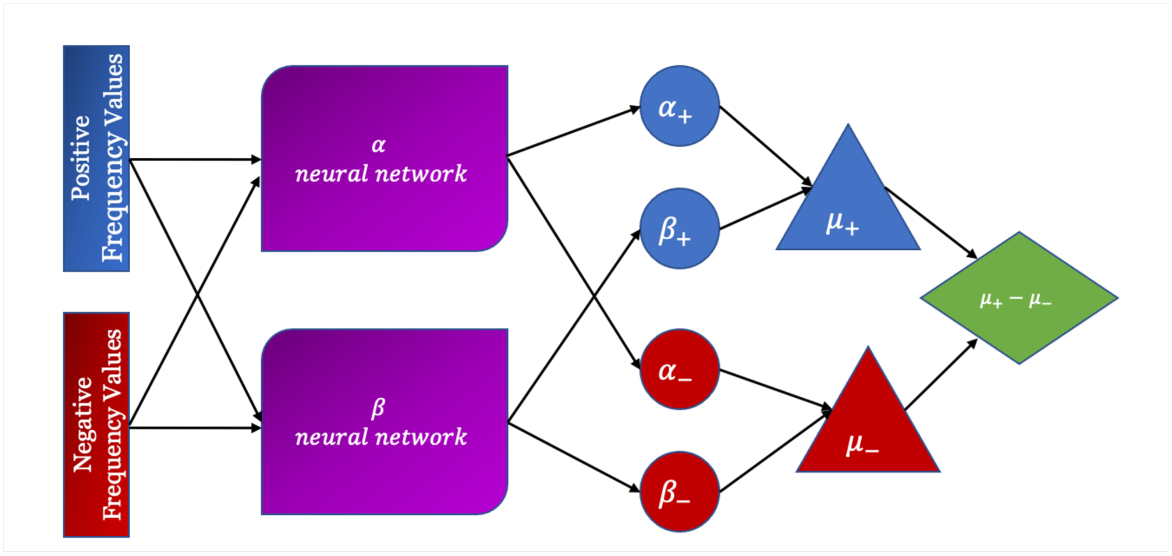
$\beta_- = f_\beta(negative\_example\_counts)$

The mean of the positive example becomes:

$$Positive\_mean = \frac{\alpha_+}{\alpha_+ + \beta_+} \tag{3.3}$$

Correspondingly, the mean of the negative example becomes:

$$Negative\_mean = \frac{\alpha_-}{\alpha_- + \beta_-} \tag{3.4}$$



Overall training architecture

Figure 3.5: Proposed Neural Network Architecture

The goal is to maximize the following value:

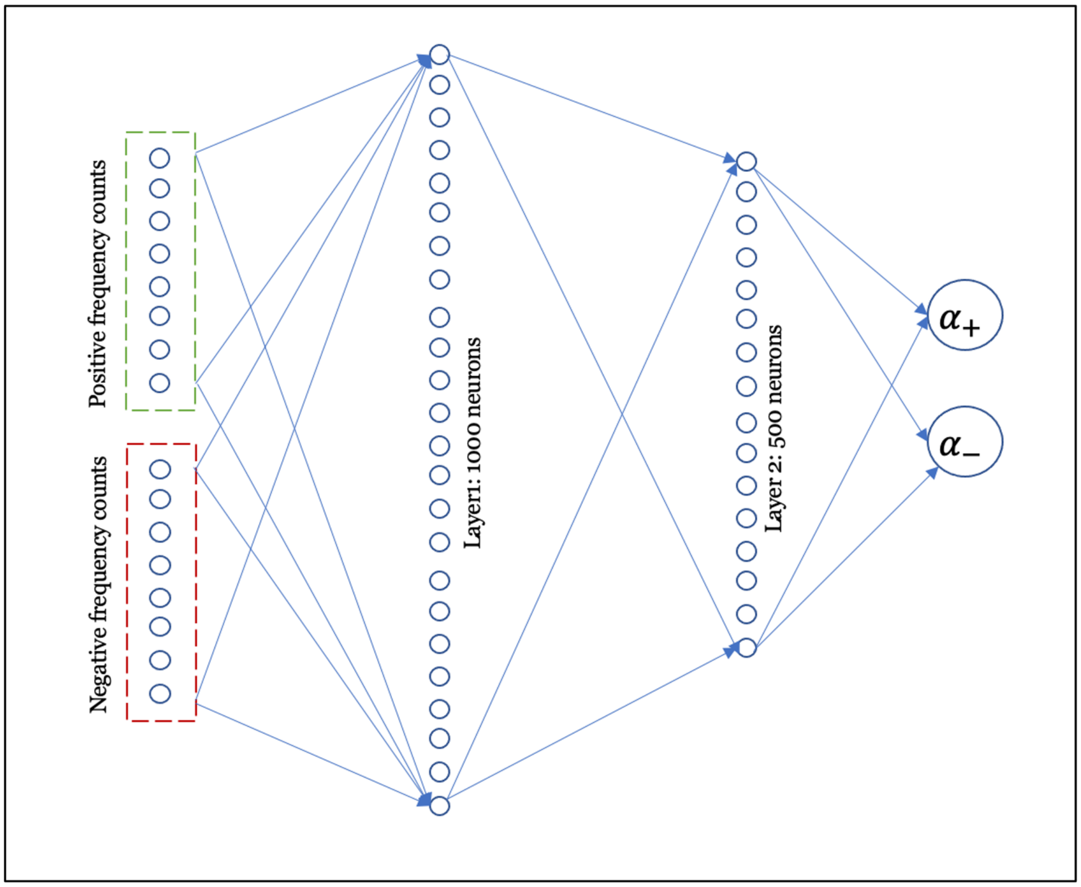
$$(Positive\_mean - Negative\_mean) \tag{3.5}$$

We employ neural networks as the functions for the estimation of alpha and beta values as shown in the above figure. Neural networks are known to be very effective at the task of function approximation. We use the feed-forward fully connected neural network both our alpha and beta functions. Each of them consist of an input layer with 8 neurons corresponding to the frequency counts, and two intermediate layers with 1000 and 500 neurons respectively.

3.4 Experimental Setup

3.4.1 Dataset

Spotify, a popular music streaming service, released a dataset for the RecSys 2018 Challenge [CLS18]. The dataset consists of a collection of playlists whereby, a playlist is an ordered sequence of songs. Playlists can be of varying lengths and a song can occur multiple times within



Neural Network Architecture for α estimation

Figure 3.6: Detailed Alpha/Beta Function

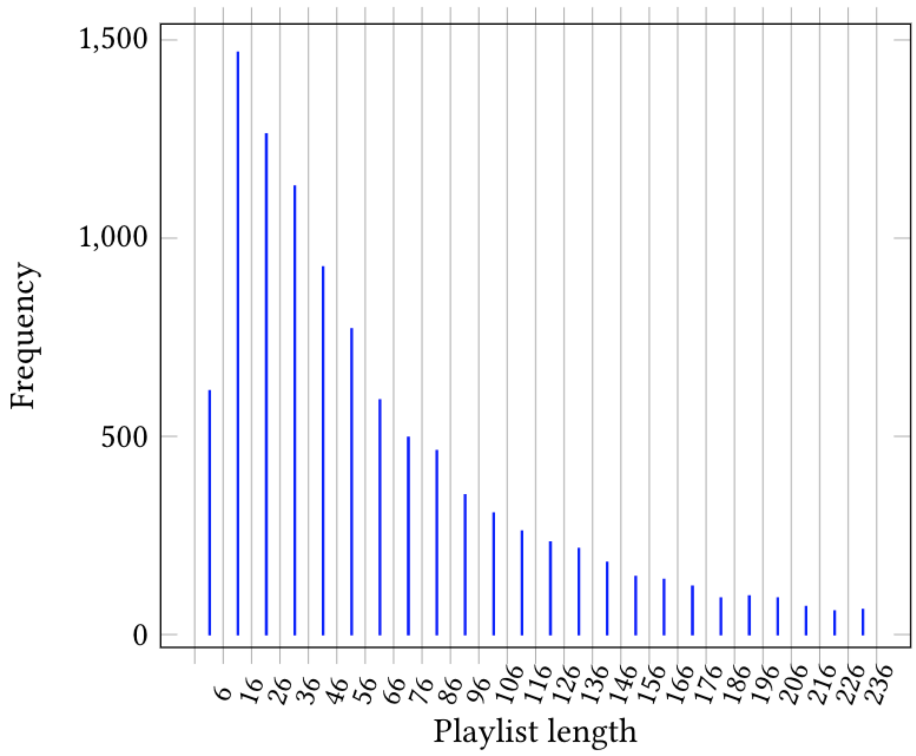


Figure 3.7: Playlist Length Frequency Distribution

a single playlist. The dataset is composed of 100,000 playlists. The distribution of playlist length demonstrates playlists tend to have less than 50 songs total. The most common playlist length is 20 songs, with its frequency being approximately 1.5% of the training dataset. There are 686,685 unique songs among these playlists. Although songs can occur multiple times, 676,244 of the unique 686,685, a vast majority, occurred less than 100 times across the entire dataset. A more detailed overview of song distribution can be found in Table 1.

The alpha and beta functions are neural networks with the following layout:

8 -> 1000 -> 500 -> 1

The input layer is a vector of 8 scalar values which represent the various frequency counts described earlier for a given sequence ‘a, b, c, d’. The 1000-neuron layer after the input layer has an exponential activation function.

The final output of the network is a scalar with an exponential activation function.

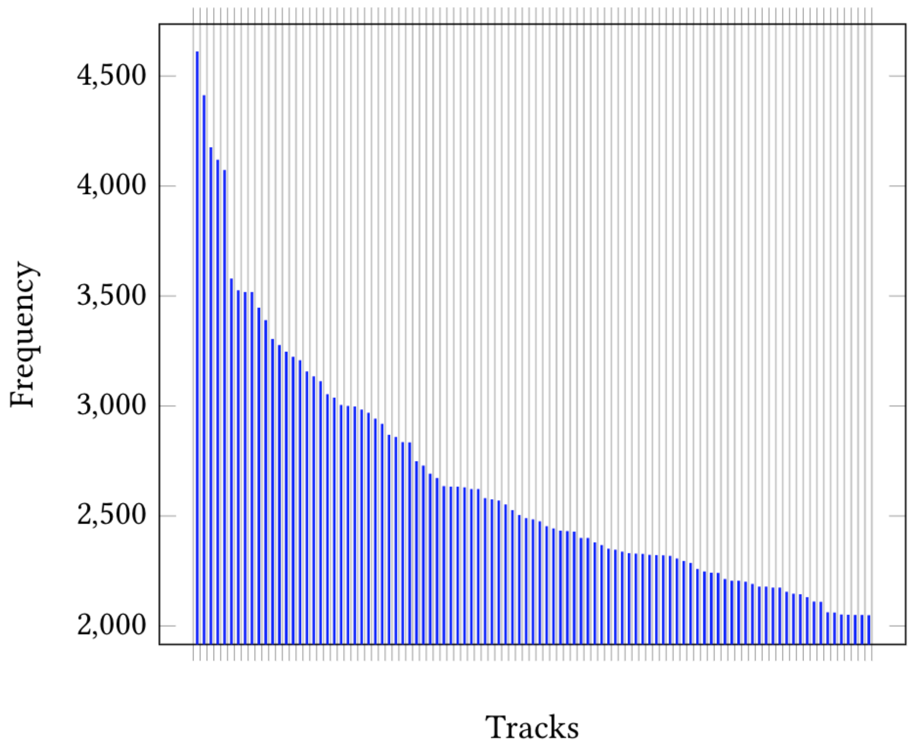


Figure 3.8: Track frequency distribution across playlists

3.4.2 Training Process

Given a train file with a list of playlists, the first step involves collecting statistics of various gram data. We utilize a Trie datastructure to store the frequencies of 1,2,3 4 gram song sequences across the playlists. The Trie [BR03] datastructure enables efficient access of the counts via the prefix notation. Trie is a tree-based datastructure whereby a node can have multiple children and the edges represent an item. The node in a Trie captures a sequence of items and can be used to store relevant information, in our case the frequency of a particular sequence so far. The only bottleneck of Trie data structure is insertion time complexity. In our setting, we perform insertion only once and the Trie offers quick retrieval of frequency count information for various sequences of items. The benefit of using a Trie over a HashTable is that Trie takes advantage of the prefix structure of sequences and hence requires significantly less memory compared to a HashTable based implementation.

Once the Trie datastructure has been populated, we need to generate the training file which is the input to our neural network. For each playlist, we go over a sequence of songs with a sliding window of size 4. This window of 4 songs would be considered as a positive instance since we have observed it in our training dataset. Given our positive instance ['a', 'b', 'c', 'd'], we generate a corresponding negative instance with the same prefix, ['a', 'b', 'c', 'e']. The negative instance is chosen based on the prefix ['a', 'b', 'c']. Item 'e' is chosen to be the most popular item given the prefix ['a', 'b', 'c']. If 'd' itself is the most popular item then 'e' is chosen to be second-most popular item given the prefix after 'd'.

This positive and negative instance pair become one training example for the neural network. Each instance is represented using the 8 count statistics described earlier. We place the counts of the instance pair adjacent to each other these set of 16 values becomes 1 training instance.

By using a sliding window approach we are able to generate multiple training instances per playlist and store all of them in the training file.

The hinge loss is used since our goal is simply to maximize the difference between the posterior mean of the positive and its corresponding negative example. Hinge loss [GW99] is defined as follows:

$$max(0, 1 - y\_true * y\_pred) \tag{3.6}$$

In our setting, the y_true value is not irrelevant and is set to a fixed scalar value for all training examples. The benefit of using hinge loss is that it only penalizes when the predicted value is opposite of the ground truth and loss remains 0 otherwise. In our case, when the y_pred value which is the difference between positive and negative instance is above 0, then it doesn't contribute to the loss. Meanwhile, when our network predicts a negative value, the loss becomes (1-y_true*y_pred) which becomes a positive value. The exact loss value depends on the magnitude of the difference in the mean did the network predict. Backpropagation [CR95] ensures that weights across the network are modified in order to reduce the loss value inturn, maximizing the difference between posterior means of positive and negative examples.

3.4.3 Candidate Generation

To improve efficiency, the underlying frequency count information is stored in a Trie data structure. The frequency counts are dependent on the training dataset and once computed does not change. Hence, the Trie data structure allows for efficient retrieval count given a prefix which is essential operation for most of our computation.

During testing, given a set of candidates 'A, B, C' we generate a set of candidates as follows: Generate the set of items that appear followed by 'A, B, C', calling it S. We then sort the items in set S in decreasing order of frequency and then select a predefined top-k items. We then rank the items in the candidate set based on our estimated posterior mean and select the item with the highest mean.

3.4.4 Evaluation Metrics

We evaluate our approach using the standard metrics used in next item prediction, namely:

- Overall Accuracy

  This metric computes the percentage of next item predictions by the model that have been accurate in the test set.

- Recall@20

  Recall is defined as the fraction of relevant items that have been retrieved.

  $$Recall@20 = \frac{|retrieved@20 \cap relevant@20|}{|relevant@20|} \tag{3.7}$$

  Recall@20 restricts the analysis to the first 20 relevant and retrieved items, whereby the order information is not taken into account. A higher Recall@20 value indicates greater overlap between the ground truth and next items predicted by the model.

- Mean Reciprocal Rank

Mean Reciprocal Rank (MRR) measures the average reciprocal rank of the ground truth next item across users in the test set.

$$MRR = \frac{1}{|U|}\sum_{u}\frac{1}{rank(u,i)} \tag{3.8}$$

Intuitively, lower the rank value of the ground truth next item, higher the MRR value indicating better performance.

3.4.5 Baselines

To better understand the results of our approach in the domain of next-item prediction, we compared the results of several other popular approaches to our approach on our Spotify dataset. Not only do these baselines include popular approaches towards next-item prediction using modern machine learning models, but our baselines also include simplistic approaches such as recommending the most popular song. The details of each baseline used for comparison can be found below.

Our baseline is to use the most popular item observed from the training data given a prefix of 3 songs. This most popular baseline is easy to implement and surprisingly performs very well on the test data.

3.4.5.1 Nearest Neighborhood

As explained by Kelen et al., the nearest neighborhood approach performs quite well for playlist continuation [KBB18] and was previously utilized on the Spotify dataset. In this approach, a playlist-track matrix is created and then utilized to create a playlist-based neighborhood model that is then used for playlist continuation. In the original implementation of the paper, the neighborhood model extends the playlist with 500 additional songs, but as a baseline we utilize the first song recommended as the sixth song given the first five seeds.

3.4.5.2 Nearest Embedding

In this approach, an embedding for every song in the dataset is computed. We employ techniques such as Word2Vec [MSC13], which look at the context information across a sliding window in playlists to place songs in an multi-dimensional embedding space.

In our baseline, we utilize a vector with dimension 100 for each song's embedding. A sliding window approach is utilized the generate the embeddings for every song in the dataset. For prediction, given a prefix, this approach simply computes the nearest song to the last song in the embedding space. The baseline presented predicts the song that maximizes the cosine similarity of the embedding vector with the fifth song. Variations to this approach includes computing an aggregate embedding of the prefix by taking average of the individual song embeddings.

3.4.5.3 Neural Sequence Learning

These types of approaches aim to learn directly from the sequences using recurrent neural network architectures. Recurrent neural networks (RNN) are extremely popular in the domain of next-item prediction, mostly due to their high performance when modeling sequential data. As described by Zhu et al.zhu2017next, RNN operates on the principle that if item A has previously been seen in a sequence then items that are very similar to item A will be seen later on in the sequence [ZLL17].

LSTM (Long Short-Term Memory) is an example of such recurrent neural network that aims to take in a sequence of items and predict the next likely item. As part of our baseline, we utilize Keras LSTM to create an RNN model for the dataset. Due to memory constrictions, the model uses the most popular 25,000 songs as a vocabulary when building the RNN. During prediction, the model takes the first five songs of the test playlist as a seed, and computes the probability of every song in the vocabulary as the sixth song given the seed. The model then recommends the song with the highest probability.

3.4.5.4 Transformer

Behaviour Sequence Transformer [CZL19] proposed by Chen et al. demonstrate the use of Transformers in a production recommender system environment. They utilize the Transformer (cite the original paper) based architecture which inherently captures the position of items across sequences and employs the self-attention mechanism. BERT (Bidirectional Encoder Represetations from Transformers) is an increasingly popular transformer-based model primarily used in the domain of natural language processing. As described by Devlin et al., BERT [DCL18], when used for language, captures information from both the left and right side of a sentence. It then performs very well on tasks such as question answering and text completion. For our baseline, we treat each playlist as a sentence with each song as a "word" in the sentence.

Our baseline utilizes DistilBERT [SDC19] that provides a smaller model compared to BERT but preserves language understanding capabilities. We provide our DistilBERT model with a vocabulary of the most popular 25,000 songs, identical to the vocabulary of the Neural Sequence Learning model proposed earlier. The DistilBERT model is evaluated using the text-completion strategy where the first five songs are given to the model as a seed and the next song is predicted by the model.

3.4.5.5 Markov Chains

Markov Chains (MC) are one of the most popular approaches towards next-item prediction. As described by Rendle et al. [RFS10], MC methods make item prediction by learning a transition graph over a sequence of items. This transition graph is then used to make further predictions based on the current items seen. In an n-th order Markov chain, the n most recent items are used as the seed to make the prediction for the next item. For our baseline, we used n-th order Markov chain with n $\in \{1, 2, 3, 4, 5\}$. During training, the transition graph is constructed by creating a graph of sequences $a_1, a_2, ..., a_n \rightarrow a_{n+1}$. During prediction, the model recommends the graph path that occurred with highest frequency using the last n songs as the sequence. In our baseline, if no sequence $a_1, a_2..., a_n$ can be found during prediction, the model will retry using the $(n-1)$-order Markov model until $n = 1$. The MC approach used as a baseline is the 2-order MC. Variations to this approach include adding weights to each item in the graph sequence.

3.4.5.6 Max Approach

The max approach is a simple intuitive, yet a very strong baseline in our next item prediction task. Given a test instance prefix the max approach recommends the most popular item seen after the prefix in the training dataset. There are variations to the max approach which are described in the results section below.

3.4.5.7 Overall Most Popular

The overall most popular approach is a very simple baseline which merely predicts the same song, the most popular song seen in the training dataset, for every single example in the testing dataset. The most popular song was HUMBLE by Kendrick Lamar, occurring 4,608 times in the training dataset.

3.5 Results

Table 3.3: Results for the Spotify Dataset

Approach	Accuracy	MRR@20	Recall@20
Bayesian	8.1%	0.11664	0.0522
Nearest Embedding	1.8%	0.0391	0.0277
Markov Chain	6.8%	0.0937	0.0345
Context Max	7.4%	0.10274	0.045975
Overall Most Popular	0.7%	0.0027	0.0105
SSE-PT	7.4%	0.1656	0.0268
BERT4Rec	2.7%	0.0773	0.0274

Approach	Accuracy	MRR@20	Recall@20
Bayesian	2.8%	0.048	0.036
Nearest Embedding	2.1%	0.0429	0.0321
Markov Chain	1.9%	0.0317	0.0103
Context Max	2.1%	0.0368	0.0419
Overall Most Popular	0.46%	0.0105	0.0347
SSE-PT	1.2%	0.0358	0.0433
BERT4Rec	1.8%	0.0453	0.0175

Table 3.4: Results for the Movielens dataset

3.6 Discussion

Our approach is novel because we combine the best of both worlds, namely: Bayesian and Neural Network approaches. The Bayesian approach we use to model the posterior mean for a particular sequence of items represents an intuitive approach of modeling using the training data.

The neural network is used as a tool to provide an accurate estimate of the alpha and beta values used in the estimation of the posterior mean. The neural network approaches are very good black-box approximation functions that can be leveraged to analyze for patterns across large amounts of training data.

Evidently, our approach performs very well compared to multiple other baselines. Surprisingly, the seeded max approach attained an accuracy of 7.4% which is a significant improvement from the unseeded overall most popular approach which attained an accuracy of 0.7%. The 2-order Markov Chain produced an accuracy of 6.8% which was the third highest accuracy. Although both neural sequence learning and transformer-based learning had relatively low accuracy, these approaches were both constrained by memory. Still, our approach is both time and space efficient while attaining excellent results.

We believe we are the first to leverage the benefits of both types of machine learning approaches and applied it to the domain of recommender systems.

3.7 Conclusion

In this chapter, we demonstrated a Bayesian approach to the next item prediction problem, popular in the recommender system domain. We have employed the benefits of neural networks in tasks where they are known to perform very well, which is function approximation given a large amounts of data. The main issue we tackle is that we are breaking the link between users and the items they consume on recommender system platform. This anonymity simulates the "incognito" behavior in recommender systems.

We believe our work is novel and among the first to combine the benefits of both Bayesian and statistical neural network based learning. The Bayesian approach is used to estimate the distribution instead of modeling the raw point estimate since a low frequency count does not provide a confident estimate of the underlying next-item. Our approach alleviates the privacy concerns of existing recommender systems since we do not track a user based on their entire history of interactions. We only focus on the immediate context when deciding the next item to recommend.

We have shown that our approach in addition to being more intuitive, also outperforms existing state-of-the-art approaches in performance. In this chapter, we have demonstrated an approach that requires users to trust the central recommender system that it will not profile the user across various sessions as currently done by existing status quo recommender systems.

CHAPTER 4

Selective Privacy Preserving Collaborative Filtering

Most industrial recommender systems rely on the popular collaborative filtering (CF) technique for providing personalized recommendations to its users. However, the very nature of CF is adversarial to the idea of user privacy, because users need to share their preferences with others in order to be grouped with like-minded people and receive accurate recommendations. Prior related work have proposed to preserve user privacy in a CF framework through different means like (i) random data obfuscation using differential privacy techniques, (ii) relying on decentralized trusted peer networks, or (iii) by adopting secured cryptographic strategies. While these approaches have been successful inasmuch as they concealed user preference information to some extent from a centralized recommender system, they have also, nevertheless, incurred significant trade-offs in terms of privacy, scalability, and accuracy. They are also vulnerable to privacy breaches by malicious actors. In light of these observations, we propose a novel *selective privacy preserving (SP2)* paradigm that allows users to custom define the scope and extent of their individual privacies, by marking their personal ratings as either *public* (which can be shared) or *private* (which are never shared and stored only on the user device). Our SP2 framework works in two steps: (i) First, it builds an initial recommendation model based on the sum of all *public ratings* that have been shared by users and (ii) then, this public model is fine-tuned *on each user's device* based on the user *private* ratings, thus eventually learning a more accurate model.

Furthermore, in this work, we introduce three different algorithms for implementing an end-to-end SP2 framework that can scale effectively as the number of items increases. Our user survey shows that an overwhelming fraction of users are likely to rate much more items to improve the overall recommendations when they can control what ratings will be publicly shared with others. In addition, our experiments on two real-world dataset demonstrate that SP2 can indeed deliver better recommendations than other state-of-the-art methods, while preserving each individual user's self-defined privacy.

4.1 Introduction

Collaborative filtering (CF) based recommender systems are ubiquitously used across a wide spectrum of online applications ranging from e-commerce (e.g. Amazon) to recreation (e.g. Spotify, Netflix, Hulu, etc.) for delivering a personalized user experience [MR16]. CF techniques are broadly classified into two types – (i) classic *Nearest Neighbor* based algorithms [TPN08] and more recent *matrix factorization techniques* [KBV09], of which the latter has been more widely and predominantly adopted in industrial applications [DUM17] for building large-scale recommender models due to its superiority in terms of accuracy [KBV09] and massive scalability [OHY15, KM16, SBS13, MBY16, Xin15, LLS16]. Regardless of the underlying technique, the performance of a CF system is generally driven by the "homophilous diffusion" [Can02b] process, where users must share some of their preferences in order to identify others with similar tastes and get good recommendations from them. The performance of CF algorithms often deteriorates without such adequate information, as often observed in the classic *cold start* [VYP17] problem.

This inherent need for a user to share his/her preferences sometimes leads to serious privacy concerns. To make things more complicated, privacy is not a static concept and may greatly vary across different users, items and places. For example, different users under changing geopolitical, social and religious influences may have varying degree of reservation about explicitly sharing their ratings on sensitive items that deal with subjects like politics, religion, sexual orientation, alcoholism, substance abuse, adultery, etc. [CPW12]. Overall, these privacy concerns can prevent a user from explicitly rating many items, which reduces the overall performance of a CF algorithm, as compared to an ideal scenario, where everyone freely rates all the items they consume.

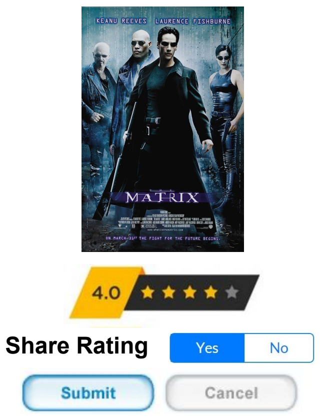


Figure 4.1: Selective privacy preserving (SP2) framework from a user's perspective.

4.1.1 Motivation

In this paper, we explore the idea of letting each user define his/her own privacy. In other words, here the user decides which ratings he/she can comfortably share *publicly* with others, while his/her remaining ratings are considered as *private*, which means that they are stored only on the user's device locally and are never shared with anyone including any peers or a centralized recommender system. Thus, this scheme enables each user to selectively define his/her own privacy. Figure 4.1 shows an example of such an operational setup. By default, every rating provided by the user is considered public and can be shared with the centralized server or peers, but the user can decide to keep a particular rating privately and not to share it with anyone when the rating is provided. In this paper, we attempt to build a CF framework that preserves each user's *selective privacy* and investigate the following issues in enabling such a framework:

• How can we build a *selective privacy preserving* (**SP2**) CF model that assimilates information from two kinds of ratings – all users' *public* ratings and each user's on-device *private* ratings?

• How can we ensure that there is no loss of *private* information in our SP2 framework?

• Can the SP2 framework improve the performance of a CF algorithm? In other words, does the SP2 framework improve the overall recommendation quality at all by taking into account each user's private ratings? Or should the users simply hold back from rating sensitive materials if they have any privacy concern?

• Can this SP2 CF model ensure scalability with respect to industrial-scale datasets?

4.1.2 Contributions

In the rest of this paper, we address the questions listed in Section 4.1.1 and make the following contributions:

• We first present the results of a user survey (Section 4.2), which shows that roughly half of users hesitate to rate or review products online due to privacy concerns and are much more likely to share more product ratings if a strong privacy guarantee is provided.

• We mathematically formulate the selective privacy preservation problem and present a formal framework to study it (Section 4.3). To the best of our knowledge, this is the first work under the umbrella of *federated machine learning* [KMR16] that supports a *private on-device* recommendation model for CF algorithms.

• We propose three different strategies (Section 4.4) for efficiently implementing an end-to-end SP2 framework, each of which is conducive to different situations. These underlying techniques overall ensure that a SP2 CF model incurs only a reasonable cost in terms of storage and communication overhead, even when dealing with massive industrial datasets or large machine learning models.

• We present analytical results on two real datasets comparing different privacy preserving and data obfuscation techniques to show the effectiveness of our SP2 framework (Section 4.5). We also empirically study what is a good information sharing strategy for any user in a SP2 framework and how much are the recommendations of a user affected, when he/she refrains from rating an item, instead of marking the latter as *private*.

4.2 User Survey: Privacy Concerns

In this section, we describe the results from a user survey. Through the survey, we tried to gauge the relevance and importance of the following two issues:

Issue 1: Is privacy an important issue for online users? Do people share fewer product ratings

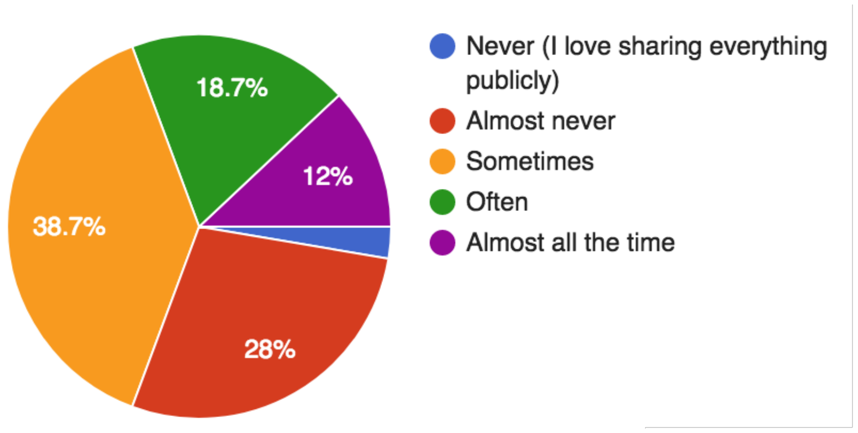


Figure 4.2: How often do you hesitate to rate an item or write a review because you do not want to share your opinion publicly or trust the platform?

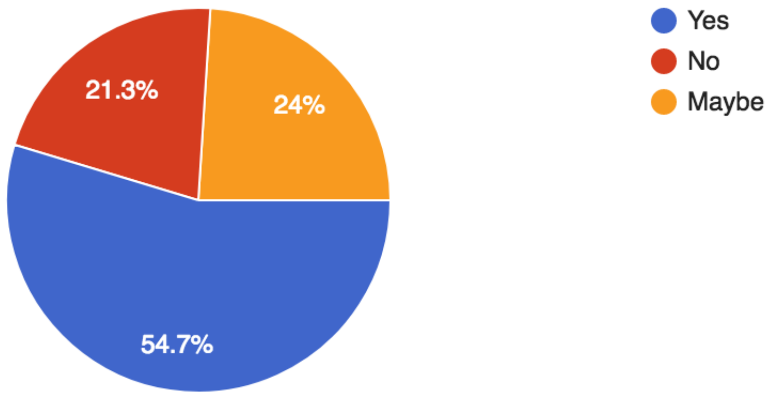


Figure 4.3: Would you rate more items if you can turn off public sharing and store your rating *privately* only on your device to improve the quality of recommendations?

because of a privacy concern?

Issue 2: If a stronger privacy guarantee is provided, are users willing to provide more product ratings so that they can get higher quality recommendations?

Our survey had 8 questions related to the issues above. The survey was given to 100 college students. A total of 75 students responded. Among them 74% were male and 24% were female. 92% of our respondents were within the age bracket $(18-30)$.

Figures 4.2 and 4.2 show the responses to two key questions in the survey related to Issues 1 and 2, respectively. In the first response, about 12% of the students stated that they almost always hesitate to rate an item because of a privacy concern. Only less than 2.7% indicated that privacy was never a concern. Overall, more than 2/3 of the respondents indicated their hesitance to rate an item due to a privacy concern. Then, when users were asked if they would rate more items if the rating is never shared with anyone else and is kept private, 56% of the users responded affirmatively (YES). Only 22% responded negatively (No) with the remaining 23% providing a tentative yes (Maybe).

In short, our survey clearly shows that there exists an unmet need from the users for a system that provides a clear and strong privacy guarantee in the context of recommendation systems. Therefore, a system like SP2 can potentially lead to higher user participation and recommendation quality. The full list of our questions and the responses are accessible at `https://goo.gl/yK2FDd` for those who are interested in the details of our survey.

4.3 SP2 Architecture

4.3.1 Data Sharing Model

Our *selective privacy preserving* (SP2) framework is based on a centralized approach. That is, (a subset of) user-provided ratings are uploaded to a centralized server, where the data is analyzed to compute items for recommendation for each user. This approach is similar to the ones used by popular online services such as Netflix and Amazon. One key difference is that SP2 allows users to explicitly mark (a subset of) their ratings as *private*, so that these ratings never leave the user's device. This option allows SP2 to provide users with an easy-to-understand-and-use privacy guarantee on their ratings. Despite this privacy guarantee, SP2 can leverage the user's private ratings to improve the quality of recommendation; when it computes recommendations, it uses each user's private ratings in addition to his/her public ratings.

Hiding (a subset of) user ratings from a central server is predicated on the following assumptions:

Assumption 1. The central recommender system is *semi-adversarial* in nature. That is, whatever data it obtains from each user, it will analyze it to the fullest extent, so that it can learn most about the user. This is not necessarily because the server has a bad intention. A server with good intentions will want to learn as much as it can about a user, so that it can provide the best possible recommendation. Despite this good will, a user may simply have a reservation on sharing certain data and want to keep it private.

Assumption 2. The central recommender system is *not malicious* in nature i.e. it will not deliberately send incorrect information to a user to adversely impact his/her recommendations. It has an incentive to provide high quality recommendations to the users.

Assumption 3. The central recommender system and the content delivery system operate in isolation from each other. This is a reasonable assumption for a large number of online recommender systems like Google Play, Yelp, Angie's List, HomeAdvisor, Glassdoor, Foursquare, etc. SP2 framework becomes highly relevant in such cases, which deal with pure recommendation sites like news portals or third party recommendation services like Yelp or app stores like Google Play, where the recommendations are solely driven by the explicit ratings from the users' community, as the external platforms do not know how a user actually interacted with the item under consideration (e.g. restaurant for Yelp and apps for Google Play Store).

It is also worth mentioning in this breath that many online services like Netflix or Hulu owns the content distribution network and as such they can always determine what items have been served to the users, even without identifying their ratings. However, as pointed out in [CPW12], the former information is less valuable as opposed to knowing the entire set of items *rated* by a user, which poses a bigger privacy concern [McC12], regardless of whether the actual rating values are known. Thus, even for online sites where the recommender system is integrated with content delivery, SP2 models can still prevent the recommender system from identifying privately rated items and the corresponding user ratings.

4.3.2 Collaborative Filtering (CF) Algorithm

The collaborative filtering (CF) algorithms used by SP2 are broadly based on the popular matrix factorization (MF) method, mainly due to its better performance, scalability and industrial applicability [KBV09, KM16, SBS13, MBY16, DUM17]. However, some of our discussions can also be extended to the traditional nearest neighbor based CF algorithms [TPN08]. We now briefly review the MF technique.

Table 4.1: Definitions of symbols used in (5.1) - (5.6)

Symbol	Definition	Symbol	Definition
μ	global mean of ratings	Ω	set of observed ratings
b_u	bias for user u	p_u	latent vector for user u
b_i	bias for item i	q_i	latent vector for item i
δ	Learning rate	λ	Regularization parameter
r_{ui}	actual rating of i by u	$\hat{r_{ui}}$	prediction of u's rating for i
e_{ui}	calculated as (r_{ui}-$\hat{r_{ui}}$)		

In the classic biased MF model [KBV09], we try to learn the latent user and item factors (assumed to be in the same feature space of dimension k) from an incomplete ratings matrix [TPN08]. More formally, here, the estimated rating for a user u on item i, $\hat{r_{ui}}$ is given by equation (5.1). The corresponding symbol definitions are provided in Table 4.1.

$$\hat{r_{ui}} = \mu + b_u + b_i + q_i^T p_u = \mu + q_i'^T p_u' \tag{4.1}$$

We compute the user and item latent factors by minimizing the regularized squared error over all the known ratings, as shown in (5.5).

$$\min \sum_{r_{ui} \in \Omega} (r_{ui} - \hat{r_{ui}})^2 + \lambda (b_i^2 + b_u^2 + \parallel q_i \parallel_2^2 + \parallel p_u \parallel_2^2) \tag{4.2}$$

This is done either using classic Alternating Least Squares method [TT12,DUM17,MBY16] which computes closed form solutions or via Stochastic Gradient Descent (SGD) [KBV09], which enjoys strong convergence guarantees [GLM16, LSJ16] and many desirable properties for scalability [OHY15, KP15]. The variable update equations for SGD are given by equation (5.6). For simplicity, we assume from now on that the user and item factors contain the respective biases i.e. user factor for u (p_u') implies the column vector $[\, b_u \quad 1 \quad p_u^T \,]^T$ and item factor for i (q_i') refers to the column vector $[\, 1 \quad b_i \quad q_i^T \,]^T$.

$$\begin{aligned}
b_u &\leftarrow b_u + \delta(e_{ui} - \lambda b_u) \\
b_i &\leftarrow b_i + \delta(e_{ui} - \lambda b_i) \\
p_u &\leftarrow p_u + \delta(e_{ui} q_i - \lambda p_u) \\
q_i &\leftarrow q_i + \delta(e_{ui} p_u - \lambda q_i)
\end{aligned} \tag{4.3}$$

4.3.3 Problem Formulation

In a SP2 framework, each user u has a set of *public* ratings, denoted by Ω^u_{public} and a set of *private* ratings, denoted by $\Omega^u_{\text{private}}$. However, since $\Omega^u_{\text{private}}$ is known only to u, the set of ratings observed here by the central recommender system is $\bigcup\limits_u \Omega^u_{\text{public}}$. We denote the latter by the notation Ω'_{public}. Now, our problem can be formulated as a *multi-objective* optimization problem, where we attempt to minimize n regularized L2 loss functions together for n users, as shown below:

min $(f_1, f_2, ..., f_n)$, where L2 loss (f_v) for user v is given by,

$$f_v : \Big[\sum_{r_{vj} \in \Omega^v_{\text{private}}} (r_{vj} - \hat{r_{vj}})^2 \Big] + \frac{1}{n} \sum_{r_{ui} \in \Omega'_{\text{public}}} (r_{ui} - \hat{r_{ui}})^2 \\ + \frac{\lambda}{n}(b_i^2 + b_u^2 + \parallel q_i \parallel_2^2 + \parallel p_u \parallel_2^2)$$

Note, traditionally multi-objective optimization problems are solved with classic techniques like *linear scalarization* (also known as the weighted sum method [GR06]). In fact, if we assign equal weights to each user's L2 loss function, then linear scalarization [GR06] can reduce this problem into a single-objective mathematical optimization problem (constructed as the weighted sum of the individual objective functions), which is similar to the one discussed in Section 4.3.2. However, due to privacy considerations, all of the data (users' ratings) cannot be pooled together; this makes classic solutions to multi-objective optimizations problems inapplicable here. We next outline a privacy-aware model to solve this problem.

4.3.4 Architecture

Given our privacy guarantee and problem formulation, one key limitation of SP2 is that the final recommendations cannot computed at the server. The server does not have a user's private ratings and is unable to incorporate these ratings in the final recommendations. Therefore, the server computes "rough estimates" of each user's preferences and each item's characteristics using the aggregated public ratings. Then each user's device downloads these user and item estimates from the server and perform the computation of the final recommendation by combining these estimates with the locally-stored user's private ratings. More formally, the following steps outline the working of our SP2 framework:

(1) The central recommender system first builds a *public* model based on all the users' shared *public* ratings. In particular, it factorizes the public ratings matrix into the user and item matrices as is shown in Figure 4.4.

(2) Each user then downloads his/her corresponding *public* user factor from the central recommender system. Additionally, all users' also download common *public item factor* data on their devices. This data is same for all users, and hence can be broadcasted by the central recommender system (for authentication in case the server cannot be trusted).

(3) Once the *public user factor* and *public item factor* are available on the device, local updates are performed on the *public user factor* using the *private* ratings, which was saved locally on the device and was never shared.

(4) The final recommendations for the user is computed using the *private* user factor and the *public item factor*. These two are stored on the user's device for future recommendations.

Figure 4.4 presents the overall architecture. Again, our framework never uploads/communicates any *private* rating, thus guaranteeing privacy preservation. This is notably different from the general federated machine learning philosophy [KMR16,BIK17]. Also note that each device has to download only one row from the public user-factor matrix since it needs the particular user's preferences not others, but it potentially has to download the entire public item factor matrix because any item can be selected for final recommendation. With millions of potential items to recom-

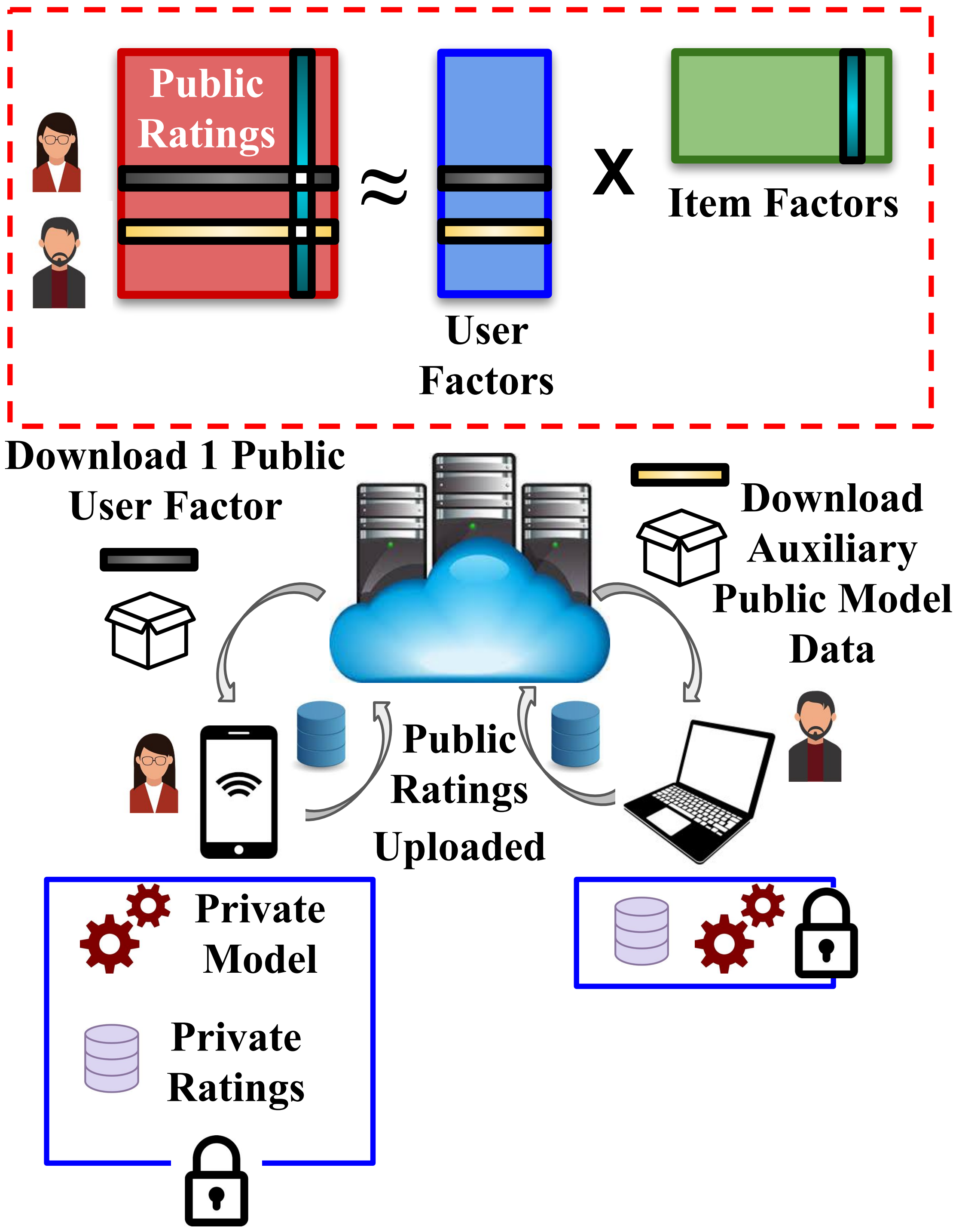


Figure 4.4: Architecture of a SP2 framework.

mend, the size of the public item factor matrix can be too large to be completely downloaded to each user's device. In the next section, we explore possible options to significant reduce the size of the item factors while we preserve the quality of the final recommendations.

4.4 Public Item Factor Matrix and CF Algorithm

In this section, we present three approaches on sending the public item factor matrix to each device and the corresponding algorithms for the updating user's latent preference vector.

4.4.1 Naive Approach

The simplest way to share the public item factor matrix is to share the entire matrix (i.e. all the latent item vectors and their biases). After the matrix factorization is performed by the recommendations provider using the public ratings from all users, we obtain the 2 latent factor matrices, namely: user factor and item factor matrix. Each user then receives their corresponding latent vector which captures their interest in a latent space of certain dimension. Each user's on-device *private* model is then built following the steps shown in algorithm . The update equation used in this algorithm are similar to the ones used in the MF model in Section 4.3.2.

algorithm

. Naive method to build on-device *private* model

```
Require: $\delta \leftarrow$ learning rate , $\lambda \leftarrow$ reg. parameter , $epochs \leftarrow$ number of epochs, $Q \leftarrow$ Aux.
    public model data containing all latent item vectors ($q_i$), item biases ($b_i$) and global ratings
    mean ($\mu$), $p_u \leftarrow$ public user latent vector for $u$, $b_u \leftarrow$ public user bias for $u$, $\Omega^u_{\text{private}} \leftarrow$
    private ratings by $u$
Ensure: $p^*_u \leftarrow$ private user latent vector for $u$, $b^*_u \leftarrow$ private user bias for $u$,
1: procedure ($\delta, \lambda, epochs, Q, p_u, b_u, \Omega^u_{\text{private}}$)
2:     $p^*_u \leftarrow p_u, b^*_u \leftarrow b_u$
3:     for $e = 0; e < epochs; e++$ do
4:         for all $r_{ui} \in \Omega^u_{\text{private}}$ do
5:             $\hat{r_{ui}} = \mu + b^*_u + b_i + q_i^T p^*_u$
```

```
6:            $e_{ui} = r_{ui}$-$\hat{r_{ui}}$
7:            $b_u^* \leftarrow b_u^* + \delta(e_{ui} - \lambda b_u^*)$
8:            $b_i \leftarrow b_i + \delta(e_{ui} - \lambda b_i)$
9:            $p_u^* \leftarrow p_u^* + \delta(e_{ui} q_i - \lambda p_u^*)$
10:           $q_i \leftarrow q_i + \delta(e_{ui} p_u^* - \lambda q_i)$
11:       **end for**
12:   **end for**
**end**
```

4.4.1.1 Top-*N* recommendation

Once the private model is built for user u, we can locally predict the rating for any item, as shown in equation (5.1), using p_u^*, b_u^*, since q_i, b_i are known for all the items as part of the public item factor matrix. These predictions can be ranked locally on the user device to provide the top-*N* recommendations.

4.4.1.2 Privacy Considerations

It is important to highlight some privacy considerations behind our naive approach:

• Even though a user only needs the corresponding item factors for each of the privately rated item to compute the on-device *private* model, the user cannot simply fetch only the desired item factors from the central recommender system since that would reveal the items that the user has rated privately. In our framework, we would like to keep not only the ratings private but also which items have been rated privately, hidden from the central recommender system.

• Consider an alternative scenario, where a user downloads only some additional irrelevant item factors to obfuscate the *private* user information. This would require downloading significantly fewer number of item factors, as compared to downloading the entire item factor matrix. However, this would make top-*N* computation infeasible locally. Now, the user needs to send back p_u^*, b_u^* to the server, which would allow the server to guess user's *private* ratings.

Similarly, sending a randomly perturbed *private* user factor back to the server can obfuscate the *private* information, but will degrade the quality of top-N recommendations.

• Consider another alternative strategy, where the actual *private* user factor is sent to the central recommender system along with multiple (k) fake user factors, thereby obfuscating the private information and making it k-anonymous [MM09a]. However, upload speeds are considerably lower than download speeds. In addition, the overall computation and communication costs can also increase by orders of magnitude, as the central servers need to compute multiple top-N recommendation lists for every user and then send all of them back.

It is important to note that the item factors matrix is downloaded only once during model building. In some situation, this does not involve unreasonable communication or storage overhead from the user end. For example, the total size (in MB) of all the item factors (I) of dimension k is given by $k \times |I| \times 8/2^{20}$, where each item factor is assumed to be an array of type `double`. Assuming $k = 100$, the download sizes for all the item factors (in raw uncompressed format) for real datasets like MovieLens [WLX16] and Netflix [WLX16] are 4MB and 10MB respectively. However, for large industrial datasets (like Amazon [MTS15]) with close to 1 million items, the raw size of all item factors (of dimension 100) grows linearly to around 763MB.

4.4.2 Clustering

We propose this method to ensure scalability of the SP2 framework as the number of items become large. One drawback with the previous mentioned naive approach is that the communication cost grows linearly with the number of items. This is because when the number of items becomes large, it results in the item latent factor matrix to be large as well. Eventually, it becomes infeasible for every user to download the entire item latent factor matrix due to expensive cost of data transfer. To mitigate this problem, we propose a clustering based approach to reduce the communication cost for users.

The intuition behind this approach is that the public item factor matrix should consist of some approximate item factors Q', which is much smaller than the set of all item factors in the original matrix Q i.e. $|Q'| < |Q|$. Now, each user u for a private rating r_{ui} should use the approximate item factor $\tilde{q}'_i$, instead of the actual item factor q'_i to compute the *private* model. This approximation introduces an error in e_{ui} calculation for each private rating r_{ui} and is given by $p_u'^*(q_i' - \tilde{q_i'})^T$, where $p_u'^*$ is the *private* user factor for u and $\tilde{q_i'} \in Q'$. Now, for each user u, we should minimize these approximation errors across all his/her *private* ratings i.e. minimize $\sum_{i \in \Omega^u_{\text{private}}} p_u'^*(q_i' - \tilde{q_i'})^T$, or $p_u'^* \sum_{i \in \Omega^u_{\text{private}}} (q_i' - \tilde{q_i'})^T$. Since, the central recommender system does not know any $\Omega^u_{\text{private}}$ for any user, the former prepares the public item factor matrix by minimizing the approximation errors across all item factors i.e. minimize $\sum_{i \in Q} (q_i' - \tilde{q_i'})^T$. This minimization goal is similar to the objective function used in clustering [KMN02]. Thus, the central recommender system performs this approximation through clustering, particularly using K-means clustering with Euclidean distance [AV07]. The individual cluster mean is treated as the approximate item factor for all the items in the cluster. In summary, the public item factor matrix for this method comprises of (1) K cluster centroids obtained after applying the K-means algorithm on all the item factors, (2) cluster membership information, which identifies which cluster an item belongs to and (3) global ratings average.

algorithm

. Building on-device *private* model via clustering

```
Require: $\delta \leftarrow$ learning rate , $\lambda \leftarrow$ regularization parameter , $epochs \leftarrow$ number of epochs, $Q' \leftarrow$
    Aux. public model data containing all cluster centers having latent vectors $(c_i)$, biases $(b_i^c)$
    and global ratings mean $(\mu)$, $C \leftarrow$ Cluster latent vectors, $\rho \leftarrow$ Cluster membership function,
    where item $i$ is mapped to cluster $\rho(i)$, $p_u \leftarrow$ public user latent vector for $u$, $b_u \leftarrow$ public
    user bias for $u$, $\Omega^u_{\text{private}} \leftarrow$ private ratings by $u$
Ensure: $p_u^* \leftarrow$ private user latent vector for $u$, $b_u^* \leftarrow$ private user bias for $u$,
 1: procedure $(\delta, \lambda, epochs, Q', p_u, b_u, \Omega^u_{\text{private}})$
 2:     if $T = \emptyset$ then
 3:         return                                                    ▷ base condition
 4:     $p_u^* \leftarrow p_u, b_u^* \leftarrow b_u$
 5:     for each cluster c do
 6:         $N_c$ = Calculate no. of items in $c$ from $\rho$
 7:     end for
```

```
8:      for $e = 0; e < epochs; e++$ do
9:          for all $r_{ui}$ in $private\_ratings_u$ do
10:             $\hat{r_{ui}} = \mu + b_u + b^c_{\rho(i)} + c^T_{\rho(i)} p^*_u$
11:             $e_{ui} = r_{ui}$-$\hat{r_{ui}}$
12:             $b^*_u \leftarrow b^*_u + \delta(e_{ui} - \lambda b^*_u)$
13:             $b^c_{\rho(i)} \leftarrow b^c_{\rho(i)} + \delta(e_{ui} - \lambda b^c_{\rho(i)})/N_{\rho(i)}$
14:             $p^*_u \leftarrow p^*_u + \delta(e_{ui} c_{\rho(i)} - \lambda p^*_u)$
15:             $c_{\rho(i)} \leftarrow c_{\rho(i)} + \delta(e_{ui} p^*_u - \lambda c_{\rho(i)})/N_{\rho(i)}$
16:         end for
17:     end for
end
```

Note, the cluster membership information for a set of I items would require $4 \times |I|$ bytes, assuming each cluster id is an integer which takes 4 bytes. For K clusters, this membership information size can be further reduced drastically using K bloom filters [Blo70, MB97] where each bloom filter represents a cluster. To demonstrate the reduction in communication costs, let consider there are 10 million items in the system. In the naive approach, if the number of latent dimensions is 100, then the total number of floating point numbers in the item latent factor matrix will be 1 billion (100*10 million). In the clustering approach, users only need to download the cluster assignment information for every item and the cluster centers. If the number of clusters is k, then the total number of floating point numbers users need to download becomes: $10\ million + 100 * k$. The number of clusters is generally chosen to be much smaller than the number of items. In this case, the clustering approach provides approximately 100x reduction in amount of data being transferred from the server to the users. It is important to note that various compression techniques can be applied on top of the 2 proposed approaches to reduce the communication cost further. The gains in reduction will be symmetric.

4.4.2.1 Top-N recommendation

In this method recall that all the item factors are not available locally on the user device. Therefore, we pursue a different strategy here: user u requests the *public* item factors for top-N' recommended items $(N' > N)$ from the central recommender system. The latter computes this using $u's$ *public* user factor (p'_u) and then sends the top N' items and their corresponding *public* item factors to u. u can re-rank these N' items based on his/her *private* user factor (p^*_u) and then select the top-N. Note, this top-N' computation by the central servers is not a privacy threat, as it can be easily calculated without any information about user's private ratings. Also, recall our assumption 2 in Section 4.3.4, which ensures that incorrect top-N' information will not be sent by the central servers.

4.4.3 Joint Optimization

Our previous approach was based on hard assignment, where each item was assigned to only one cluster. However, soft clustering techniques like non-negative matrix factorization (NMF) [LS00] considers each point as a weighted sum of different cluster centers. In this approach, we try to perform soft clustering on all the item factors simultaneously as the *public* recommendation model is built. In other words, the central recommender system jointly learns the *public* model and the soft cluster assignments. For this, we revise the equations (5.1) and (5.5) to (4.4) and (4.5), where C denotes the cluster center matrix of dimension $k \times z$ (z being the number of clusters), and w_i is a column vector representing the different cluster weights (non-negative) for item i. This problem can be formulated as a constrained optimization problem and algorithm shows how the central recommender system performs this joint optimization. One key aspect in this algorithm is that the weights are updated (step 14) using projected gradient descent (PGD) [Lin07], in order to ensure that all cluster weights are non-negative. This facilitates in finding the top-R cluster assignments for any item by finding the highest R corresponding weights. Finally, the public item model data for this approach should consist of the following: (1) the cluster center matrix C, (2) item biases b_i, (3) top-R cluster weights (in descending order) for each item i, the corresponding cluster ids and (4) the global ratings mean. Using C and top-R cluster weights for any item i, user u can locally approximate the *public* item factor for any item by its weighted sum of top-R cluster centers i.e. $\sum_{n \in \text{top } R} w_n C_n$ (C_n represents the n^{th} cluster center). With this approximation, u can now use algorithm to compute the on-device *private* model again. Note, when R is small, we can save a significant communication cost by sending only top-R weights as compared to the naive approach.

$$\hat{r_{ui}} = \mu + b_u + b_i + w_i^T C^T p_u \tag{4.4}$$

$$\min. \sum_{r_{ui} \in \Omega'_{\text{public}}} (r_{ui} - \hat{r_{ui}})^2 + \lambda (b_i^2 + b_u^2 + \parallel w_i \parallel_2^2 + \parallel c \parallel_2^2 + \parallel p_u \parallel_2^2) \qquad \text{s.t.} w_{ij} \geq 0. \tag{4.5}$$

algorithm

. Joint optimization based matrix factorization

```
Require: $\delta \leftarrow$ learning rate , $\lambda \leftarrow$ regularization parameter , $epochs \leftarrow$ number of epochs,
    $\Omega'_{\text{public}} \leftarrow setofallpublicratings$
Ensure: $C, p_u, b_u, b_i, w_i$ for all users and items
 1: procedure $(\delta, \lambda, epochs, \Omega'_{\text{public}})$
 2:     $\mu = \text{Mean}(\Omega'_{\text{public}})$
 3:     Initialize $b_u, p_u, b_i, w_i, C$ with values from $N(0, 0.01)$.
 4:     for $e = 0; e < epochs; e++$ do
 5:         for all $r_{ui}$ in $private\_ratings_u$ do
 6:             $\hat{r_{ui}} = \mu + b_u + b_i + w_i^T C^T p_u$
 7:             $e_{ui} = r_{ui}$-$\hat{r_{ui}}$
 8:             $b_u \leftarrow b_u + \delta(e_{ui} - \lambda b_u)$
 9:             $b_i \leftarrow b_i + \delta(e_{ui} - \lambda b_i)$
10:             $p_u \leftarrow p_u + \delta(e_{ui} c w_i - \lambda p_u)$
11:             $C \leftarrow C + \delta(e_{ui} w_i p_u^T - \lambda c)$
12:             $w_i \leftarrow w_i + \delta(e_{ui} c^T p_u - \lambda w_i)$
13:             for each $w \in w_i$ do
```

```
14:                 $w \leftarrow \mathrm{Max}(w, 0)$ //PGD
15:             **end for**
16:         **end for**
17:     **end for**
end
```

4.4.3.1 Top-*N* recommendation

Interestingly, with the public item model data for this method, user *u* can locally compute the approximation for each item factor, as mentioned above. As a consequence, *u* is also able to locally compute the top-*N* recommendations using these approximate item factors.

4.5 Experiments

We compared the performance of our SP2 framework with various baselines, as described next, under different settings on two real datasets, viz., MovieLens-100K [HK15] data and a subset of Amazon Electronics [MTS15] data. We outperform existing state-of-the-art baselines in effectiveness on both metrics, namely: Root Mean Squared Error (RMSE) and Normalized Discounted Cumulative Gain (nDCG). Our SP2 framework performs close to the optimal approach in terms of effectiveness which is to not allow users to rate privately and keep all ratings as public.

4.5.1 SP2 vs. Different Baselines

• *Absolute Liberal (Everything public)*: Here, we assume that every user liberally shares everything publicly without any privacy concern i.e. a single MF model is built on the entire training data itself. Theoretically, this should have the best performance, thus providing the overall upper bound.

• *Absolute Conservative (Everything private)*: Here, we assume that every user is conservative and does not share anything publicly due to privacy concerns. Thus separate models are built for each user based *only* on their individual ratings, which in practice, is as good as using the average rating for that user for all his/her predictions.

• *Only Public*: This mimics the standard CF scenario, where privacy preserving mechanisms are absent. Consequently, the users only rate the items, which they are comfortable with sharing; they refrain from explicitly rating sensitive items. We build a single MF model using **only** the *public* ratings and ignore the *private* ratings.

• *Distributed aggregation*: Shokri et al. [SPT09a] proposed three peer-to-peer based data obfuscation policies, which obscured the user ratings information before uploading it to a central server: (1) *Fixed Random (FR) Selection*: A fixed set of ratings are randomly selected from other peers for obfuscation. (2) *Similarity-based Random (SR) Selection*: A peer randomly sends a fraction of its ratings to the user for obfuscation depending on its similarity with the user. (3) *Similarity-based Minimum Rating (SM) Frequency Selection*: This is similar to the SR policy, except that during selection higher preference is given to those items that have been rated the least number of times.

• *Fully decentralized recommendation*: Berkovsky et al. [BEK07a] proposed a fully decentralized peer-to-peer based architecture, where each user requests rating for an item by exposing a part of his/her ratings to a few trusted peers. The peers obfuscate their profiles by generating fake ratings and then compute their profile similarities with the user.

• *Differential Privacy*: McSherry et al. in [MM09a] masks the ratings matrix sufficiently by adding random noise, drawn from a normal distribution, to generate a noisy global average rating for each movie. These global averages are then used to generate β_m fictitious ratings to further obscure the ratings matrix.

For all MF models, the hyper-parameters were initialized with default values from the open-source `Surprise` package.

4.5.2 Private Ratings Allocation

For analyzing the efficacy of our SP2 framework, it is also important to consider how users privately rate an item. We next define two ratios to characterize this:

• *User privacy ratio* for a user u is defined as the fraction of $u's$ total ratings which are marked *private* by u.

• *Item privacy ratio* for an item i is likewise defined as how many of the total users (which assigned *i* a rating) have marked *i* as *private*.

Now, we consider two different hypotheses for modeling users' behavior in keeping their ratings private:

• **Hypothesis 1 (H1).** Users always decide independently which of his/her ratings are *private*. Formally, for any two users x and y, who have rated an item i with ratings r_{xi} and r_{yi} respectively, $P(r_{xi} \text{ is private} \mid r_{yi} \text{ is private}) = P(r_{xi} \text{ is private})$. In our experiments, we generate user privacy ratios in the interval $[0,1]$ for all n users from a beta distribution [LCG17] with parameters α, β. For each user u with user privacy ratio γ_u, $(1-\gamma_u)$ fraction of u's ratings are randomly selected and marked as *public*, while the remainder of u's ratings are considered *private*.

• **Hypothesis 2 (H2).** Users do not decide independently which of his/her ratings are *private*. In other words, ratings for some items are more likely to be marked as private. Formally, using the same mathematical notations as above, $P(r_{xi} \text{ is private} \mid r_{yi} \text{ is private}) \neq P(r_{xi} \text{ is private})$. Here, we generate item privacy ratios for all m items from a beta distribution. For each item i with item privacy ratio γ_i, $(1-\gamma_i)$ fraction of ratings assigned to i are randomly selected and marked as *public*, while the remainder of i's ratings are considered *private*.

The following parameters for the beta distribution (shown in Figure 4.5) is chosen to capture different scenarios.

1. *Mostly Balanced* $(\alpha = 2, \beta = 2)$: Most user/item privacy ratios are likely to be close to the theoretical mean value 0.5.

2. *Mostly Extreme* $(\alpha = 0.5, \beta = 0.5)$: Most users/items have either very high or very low privacy ratios. The overall average of the privacy ratios will be close to 0.5.

3. *Mostly Conservative* $(\alpha = 5, \beta = 1)$: Most users/items have very high privacy ratios.

4. *Mostly Liberal* $(\alpha = 1, \beta = 5)$: Most users/items have very low privacy ratios.

4.5.3 Evaluation Setup

We evaluate our SP2 framework using accuracy-based as well as ranking-based metric. The definition of the metrics used for evaluation are given below.

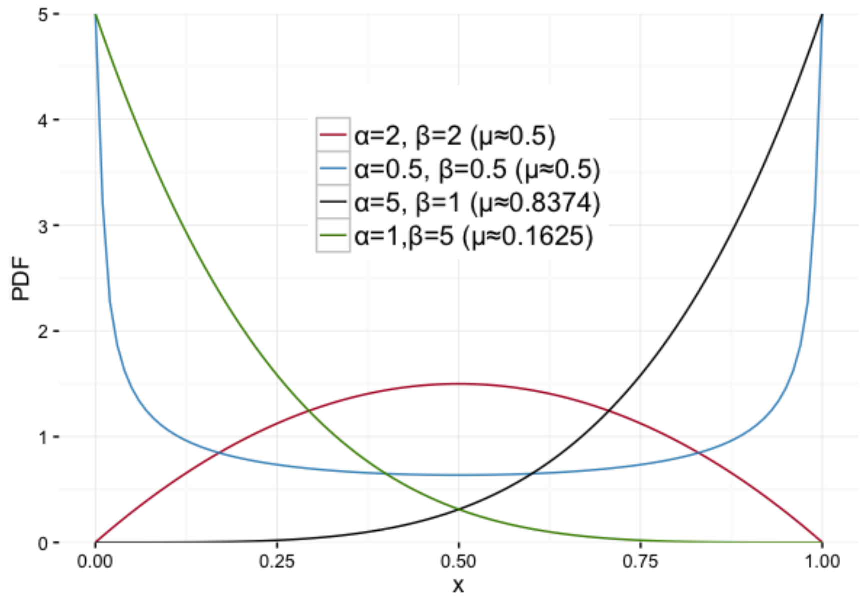


Figure 4.5: Probability density functions of four different beta distributions used in private ratings allocation.

• *Root Mean Squared Error (RMSE)*: Root Mean Squared Error is used to compute the offset between the predicted and actual rating for a particular user.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(r_{ui} - \hat{r}_{ui})^2} \tag{4.6}$$

The RMSE for the overall recommender system is calculated as the average of individual user RMSE. Lower RMSE value indicates a better recommender system performance.

• *Normalized Discounted Cumulative Gain (NDCG)*: Given a list of N recommended items, NDCG [LLL16] is used to measure the ranking quality of a recommender system. It is computed as follows:

$$NDCG@N = \frac{1}{IDCG}\sum_{i=1}^{N}\frac{2^{rel_i} - 1}{log(i+1)} \tag{4.7}$$

where IDCG represents the maximum DCG ranking value. *rel_i* represents the binary relevance of the recommended item at position i. The intuition behind a better/higher NDCG value is that relevant items should appear in earlier positions of the recommendation list. The NDCG for a recommender system is computed by taking the average over individual user NDCG values.

• *Data Sets*: We use two rating datasets for our experiments: Movielens-100k [HK15] and Amazon Electronics [MTS15].

4.5.4 Results

The 5-fold average RMSE and NDCG@10 scores [LLL16] along with their corresponding standard deviations are reported in Table 4.6 for the MovieLens and Amazon Electronics datasets. The RMSE results in Table 4.6 indicate that, as expected, *absolute liberal* performs the best (lowest RMSE), whereas *absolute conservative* performs the worst. We also observed that for varying ratios of *r* (public:private ratings), our SP2 framework consistently outperforms the standard CF framework where only public data is used. However, even in SP2, the RMSE reduces as more public ratings are available, which is intuitively true. The three P2P based CF strategies [SPT09a] do not perform well at all, due to the noise and obfuscation added to the ratings matrix and at times, even produce higher RMSE as compared to the *absolute conservative* method. Thus, our SP2 framework fits better models than the standard CF method or P2P based architecture.

As indicated by the results in Table 4.6, the peer-to-peer based techniques and the differential privacy method, which attempt to ensure *complete* user privacy from the central recommender system, end up performing worse than the standard *only public* baseline due to the data obfuscation policies. In addition, the fully decentralized approach in [BEK07a] is not scalable due to the limited number of trusted peers. In the same vein, the distributed aggregation approaches in [SPT09a] suffer from poor performance as the number of peers increases due to higher obfuscation; however, lowering the number of peers risks significant privacy breach by the central recommender system. Table 4.6 further summarizes that our joint optimization approach (with only top-3 cluster weights) performs as good as the naive approach. Our clustering approach for SP2 framework, performs worse than naive and joint optimization but is largely better than the *only public* baseline across both evaluation metrics. Unless otherwise mentioned in the table, *P*-value for all results related to SP2 framework (computed using two-tailed test with respect to *only public* baseline) is less than 0.001. As evident from the table, our results hold across both the hypotheses. However, the performance of all the implementations improve as the privacy ratio reduces. We next summarize

Category	Method	Model Parameters	Movielens		Amazon Electronics	
			RMSE	NDCG@10	RMSE	NDCG@10
Peer-to-peer Based	Shokri et al. (FR)	#Peers= 10	1.1624±0.00189	0.4873±0.0055	1.2216±0.01229	0.7757±0.00841
	Shokri et al. (SR)	#Peers= 10	1.1624±0.00562	0.4891±0.00773	1.2048±0.00889	0.7774±0.00686
	Shokri et al. (SM)	#Peers= 10	1.1447±0.00629	0.4922±0.0094	1.2028±0.00985	0.7748±0.00887
	Berkovsky et al.	#Peers= 40	1.132±0.00411	0.4876±0.00599	1.3405±0.00562	0.7619±0.00756
Diff. Privacy	McSherry et al.	$\beta_m = 15$	1.201±0.00675	0.4795±0.00911	1.1349±0.00664	0.7719±0.00675
Extreme Baselines	Abs. Conservative	$k = 100$, #epochs = 20	0.9632±0.00489	0.4132±0.00661	0.9788±0.00368	0.7379±0.00535
	Abs. Liberal	$k = 100$, #epochs = 20	0.8923±0.00576	0.5426±0.0072	0.9538±0.00955	0.788±0.00818
Classic Collaborative Filtering	Only Public (H1)	$\alpha = 2, \beta = 2, \mu = 0.48$	0.9183±0.00725	0.545±0.00726	0.971±0.00516	0.7892±0.00334
		$\alpha = 0.5, \beta = 0.5, \mu = 0.48$	0.925±0.0075	0.5468±0.00688	0.9775±0.00455	0.788±0.00204
		$\alpha = 5, \beta = 1, \mu = 0.82$	0.9518±0.00822	0.5363±0.00727	0.9957±0.00763	0.7738±0.00233
		$\alpha = 1, \beta = 5, \mu = 0.17$	0.9033±0.00641	0.5534±0.00179	0.96±0.00411	0.7955±0.00446
	Only Public (H2)	$\alpha = 2, \beta = 2, \mu = 0.48$	0.9206±0.00328	0.5391±0.00179	0.9692±0.00895	0.787±0.00463
		$\alpha = 0.5, \beta = 0.5, \mu = 0.48$	0.9287±0.00228	0.528±0.00596	0.969±0.00931	0.7853±0.00242
		$\alpha = 5, \beta = 1, \mu = 0.82$	0.9522±0.00213	0.517±0.00797	0.9851±0.00808	0.7718±0.00517
		$\alpha = 1, \beta = 5, \mu = 0.17$	0.9063±0.00294	0.5466±0.00212	0.9581±0.00946	0.7929±0.00456
Selective Privacy Preserving (SP2)	**Naive (H1)**	$\alpha = 2, \beta = 2, \mu = 0.48$	**0.9051±0.00654**	**0.5558±0.00511**	**0.9613±0.00534**	**0.7991±0.00322**
		$\alpha = 0.5, \beta = 0.5, \mu = 0.48$	**0.9072±0.00873**	**0.5542±0.00727**	**0.9641±0.00555**	**0.7978±0.0012**
		$\alpha = 5, \beta = 1, \mu = 0.82$	**0.9316±0.0088**	**0.5444±0.00666**	**0.9808±0.00733**	**0.7868±0.00297**
		$\alpha = 1, \beta = 5, \mu = 0.17$	**0.8953±0.00688**	**0.5594±0.00696**	**0.9526±0.00525**	**0.8048±0.00318**
	Naive (H2)	$\alpha = 2, \beta = 2, \mu = 0.48$	**0.907±0.00377**	**0.5514±0.00123**	**0.9589±0.00921**	**0.7977±0.00378**
		$\alpha = 0.5, \beta = 0.5, \mu = 0.48$	**0.914±0.00302**	**0.5383±0.00491**	**0.9603±0.00937**	**0.793±0.00222**
		$\alpha = 5, \beta = 1, \mu = 0.82$	**0.9316±0.00215**	**0.5274±0.00802**	**0.9705±0.00795**	**0.7824±0.00459**
		$\alpha = 1, \beta = 5, \mu = 0.17$	**0.8946±0.0032**	**0.5532±0.00306**	**0.9517±0.00949**	**0.8034±0.00411**
	Clustering (H1)	$\alpha = 2, \beta = 2, \mu = 0.48$	0.9165±0.00766	0.5457±0.00695	0.966±0.00565	0.7893±0.00353[a]
		$\alpha = 0.5, \beta = 0.5, \mu = 0.48$	0.9146±0.01183	0.5494±0.00795	0.9695±0.00774	0.7876±0.00309
		$\alpha = 5, \beta = 1, \mu = 0.82$	0.9387±0.00854	0.5366±0.00681	0.9847±0.00741	0.7736±0.00241
		$\alpha = 1, \beta = 5, \mu = 0.17$	0.9037±0.00634	0.5538±0.00206	0.958±0.0048	0.7945±0.00373
	Clustering (H2)	$\alpha = 2, \beta = 2, \mu = 0.48$	0.9183±0.00395	0.5401±0.00172	0.9653±0.00924	0.7871±0.00464[b]
		$\alpha = 0.5, \beta = 0.5, \mu = 0.48$	0.9249±0.00206	0.5287±0.00598	0.9651±0.00938	0.7852±0.00228
		$\alpha = 5, \beta = 1, \mu = 0.82$	0.9405±0.00166	0.5174±0.00718	0.9757±0.00812	0.7716±0.00528
		$\alpha = 1, \beta = 5, \mu = 0.17$	0.9047±0.00319	0.5473±0.00121	0.9566±0.00971	0.7926±0.00436
	Joint Opt. (H1)	$\alpha = 2, \beta = 2, \mu = 0.48$	**0.9051±0.00654**	**0.556±0.00502**	**0.9612±0.00533**	**0.7989±0.00315**
		$\alpha = 0.5, \beta = 0.5, \mu = 0.48$	**0.9072±0.00873**	**0.5537±0.00735**	**0.964±0.00556**	**0.7975±0.00098**
		$\alpha = 5, \beta = 1, \mu = 0.82$	**0.9316±0.0088**	**0.5447±0.00646**	**0.9808±0.00734**	**0.7869±0.00338**
		$\alpha = 1, \beta = 5, \mu = 0.17$	**0.8953±0.00689**	**0.5592±0.00706**	**0.9526±0.00524**	**0.8045±0.00318**
	Joint Opt. (H2)	$\alpha = 2, \beta = 2, \mu = 0.48$	**0.907±0.00377**	**0.551±0.00095**	**0.9589±0.0092**	**0.7978±0.00371**
		$\alpha = 0.5, \beta = 0.5, \mu = 0.48$	**0.914±0.00302**	**0.5383±0.00507**	**0.9602±0.00939**	**0.793±0.0022**
		$\alpha = 5, \beta = 1, \mu = 0.82$	**0.9319±0.00241**	**0.5278±0.00851**	**0.9705±0.00792**	**0.782±0.00479**
		$\alpha = 1, \beta = 5, \mu = 0.17$	**0.8947±0.0032**	**0.5533±0.00289**	**0.9517±0.0095**	**0.8034±0.00423**

[a] $P_{\text{value}} < 0.02$

[b] $P_{\text{value}} < 0.1$, statistically insignificant

Figure 4.6: Results of SP2

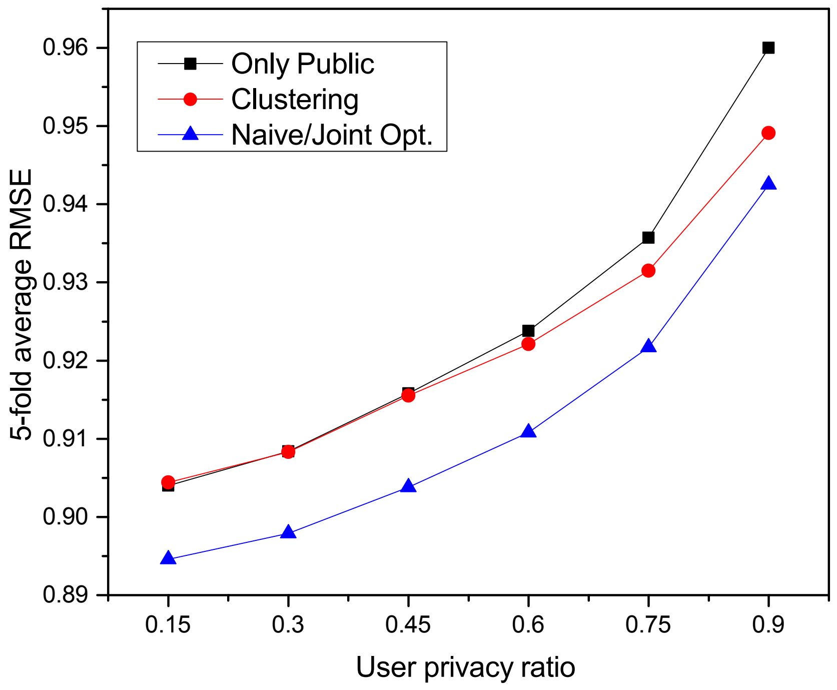


Figure 4.7: RMSE comparison

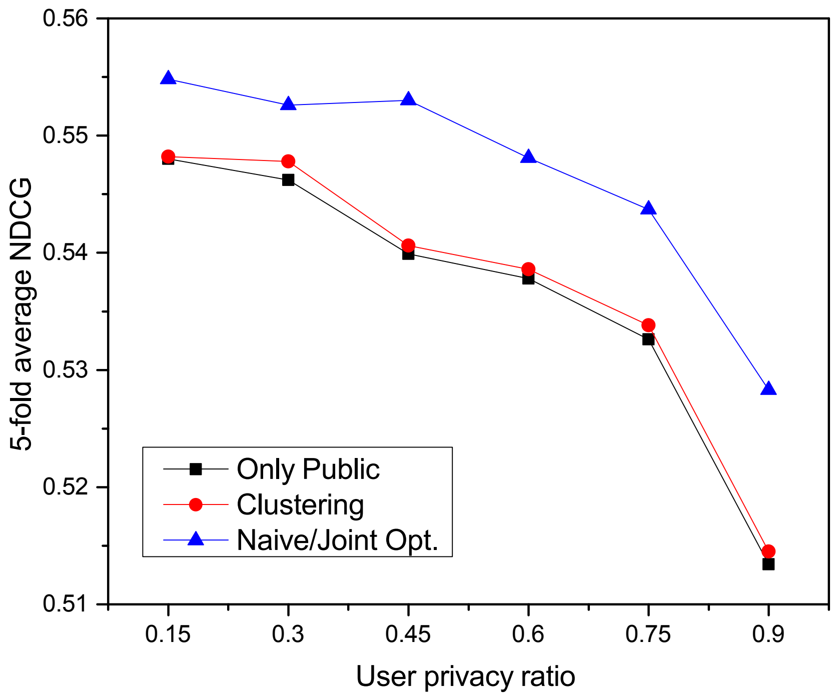


Figure 4.8: NDCG comparison

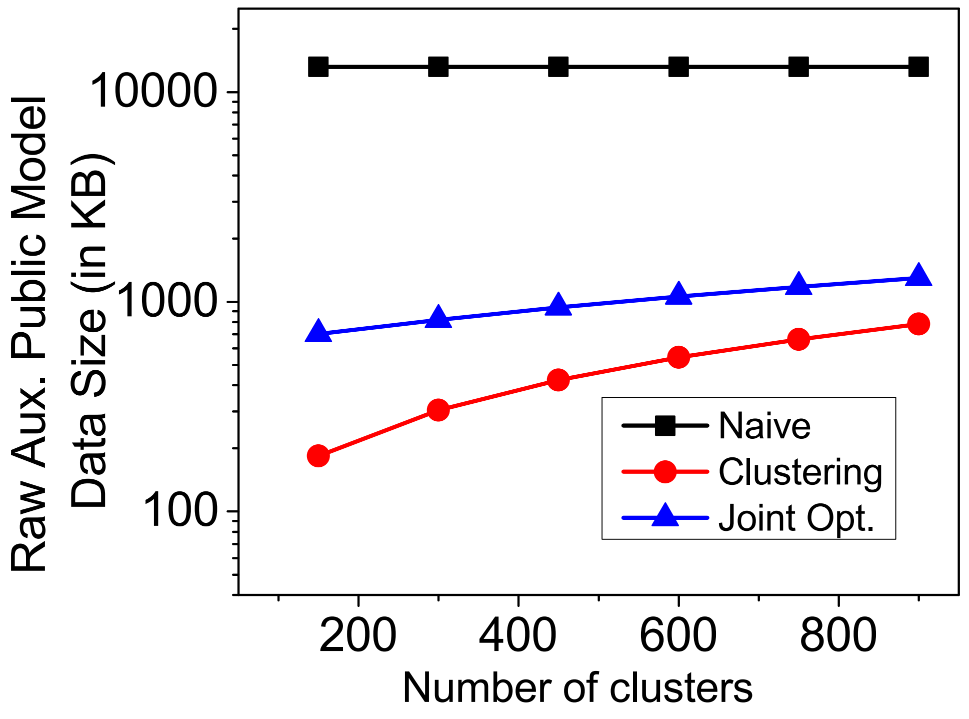


Figure 4.9: Aux. data size comparison

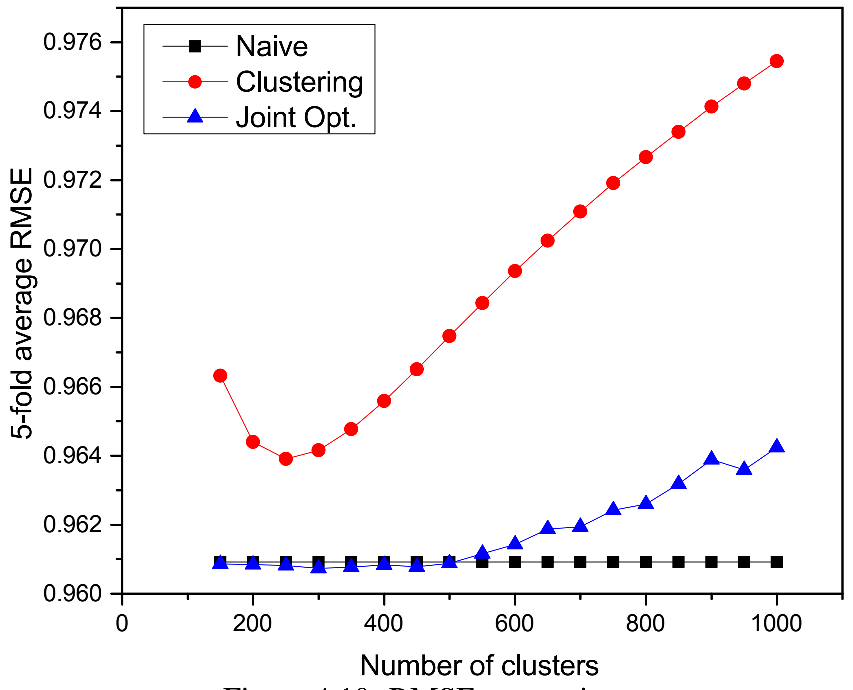


Figure 4.10: RMSE comparison

the key results below:

• *Comparison under H1*: Under Hypothesis 1 (H1), where users always decide independently which of his/her ratings are private, our proposed SP2 approach performs better than state-of-the-art approaches which are part of our baselines. Across various beta distributions used for sampling the privacy ratio of each user, SP2 performs better than the only public baseline in both metrics RMSE and NDCG. In the Mostly Conservative scenario, our SP2 joint optimization approach provides an improvement of 2.12% in RMSE performance (lower value) and 1.56% improvement in NDCG over the Only Public baseline. Our results are statistically significant with a p-value less than 0.001.

• *Comparison under H2*: Under Hypothesis 2 (H2), ratings for some items are more likely to be marked as private, our proposed SP2 approach performs better than state-of-the-art approaches. Across various beta distributions used for sampling the privacy ratio of each user, SP2 performs better than the only public baseline in both metrics RMSE and NDCG. In the Mostly Conservative scenario, our SP2 joint optimization approach provides an improvement of 2.13% in RMSE performance (lower value) and 2.08% improvement in NDCG over the Only Public baseline. Our results are statistically significant with a p-value less than 0.001.

• *Scalability*: We perform an experiment to study the scalability of our SP2 framework. As the number of items in the recommender system increases, our SP2 framework can scale accordingly by increasing the number of clusters, a parameter for the clustering and joint optimization based approaches. Firstly, we study the relationship between the number of clusters and data size of the raw auxiliary public model. As seen in Figure 4.9, the data size for the naive approach remains independent of the number of clusters, since users download the entire item latent factor matrix. Whereas for the clustering and joint optimization approaches, as expected the data size increases as the number of clusters increase but the data size is orders of magnitude less than the naive approach. We also study the relationship between the number of clusters and 5-fold average RMSE across the 3 proposed approaches for SP2. As shown in Figure 4.10, the naive approach performs the best followed by the joint optimization approach and finally the clustering approach. As the number of clusters increases the RMSE for the clustering and joint optimization initially decreases up to an optimal number of clusters and then starts to rise linearly with the number of clusters. In other words, the naive method has the best performance but requires the largest auxiliary model data. On the other hand the joint optimization technique require an order of magnitude less data than the naive one but can reach the same performance for an optimal number of clusters.

• *Ablation Study with Privacy Ratio*: In our SP2 framework, we compare the 3 proposed approaches for auxiliary public model data which needs to be downloaded by every user device. We perform an ablation study whereby we choose a fixed privacy ratio for every user and compare the 3 approaches in terms of the two specified metrics, namely RMSE and NDCG. As shown in Figure 4.7, we observe two trends. First, as the user privacy ratio increases the overall RMSE goes up as expected due to the decrease in public ratings shared by users. Second, among the three approaches Joint Optimization and Naive approaches which perform similarly have the best performance (lower RMSE), followed by the Clustering approach and the Only Public baseline which performs the worst. We observe a similar trend in terms of performance for NDCG as shown in Figure 4.8. We performed a similar ablation study considering a fixed item privacy ratio (H2) and we obtain similar results. Due to space constraints, we have decided to omit their plots.

4.6 Related Work

Privacy preserving recommender systems has been well explored in the literature. Peer-to-peer (P2P) techniques [BEK07a] are largely meant to protect users from untrusted servers. However, they also require users to share their private information with peers, which is a privacy breach in itself. In addition, P2P architectures lack scalability due to limited number of trusted peers and are vulnerable to malicious interferences by rogue actors. Differential privacy methods [MM09a] provide theoretical privacy guarantees for all users, but can also adversely impact the performance of the recommender systems due to data obfuscation.

The related literature also comprises of cryptology [ZWH08] based techniques that approach the problem little differently. For example, Zhan et al. [ZWH08] used "homomorphic encryption" to integrate multiple sources of encrypted user ratings in a privacy preserving manner. However, the extreme computation time and scalability issues associated with homomorphic encryption pose a serious practicality question [NLV11], even for moderate size datasets.

Lastly, recent federated machine learning approaches [KMR16] have proposed privacy-preserving techniques to build machine learning models using secure aggregation protocol [BIK17]. However, in case of CF algorithms, this would require a user to share an update (in encrypted form) performed on an item factor locally. In our case, this means that the server would be able to identify from the encrypted updates, which items the user had rated privately, even though the exact ratings remain unknown. This itself constitutes a serious privacy breach [CPW12, Adv87, McC12]. Hence, in our SP2 framework, no private user information is ever uploaded or communicated.

4.7 Conclusion

In this chapter, we propose a novel selective privacy preserving (SP2) paradigm for CF based recommender systems that allows users to keep a portion of their ratings *private*, meanwhile delivering better recommendations, as compared to other privacy preserving techniques. We have demonstrated the efficacy of our approach under different configurations by comparing it against other baselines on two real datasets. Finally, our framework empowers users to define their own privacy policy by determining which ratings should be *private* and which ones should be *public*. An overwhelming majority of users strongly support this policy, as have been showcased through the user survey.

CHAPTER 5

Federated Local Differential Privacy for Recommender Systems

Current industrial recommender systems pose privacy risks for users since they rely on detailed historical interaction information. These interactions can accurately capture the preferences and can paint a detailed picture of the user. To alleviate these privacy concerns, we propose a federated learning based approach that can mathematically quantify the privacy gain using local differential privacy. Our framework allows each individual user to determine their own privacy level via the ε parameter of differential privacy. We demonstrate the trade-offs between privacy and accuracy by simulating various ε values from the Beta distribution on two real-world datasets.

5.1 Introduction

Recommender Systems are ubiquitous on the internet. They are used across platforms to personalize items such as movies, songs, books, products to buy etc. Recommender Systems have become a vital component that drive user engagement and provide personalized content. For example, on the question answering site Quora [YA16], almost all features relies on recommendation systems ranging from: the ranking of answers to a question, which users to follow, which questions to show to a user on the homepage, among many others.

Recommender Systems aim to learn user interests based on historical interactions and other metadata. They essentially assist the user in decision-making since it's not scalable for a user to experience every item in the marketplace. For example, in a movie recommender system, it's impossible for a user to watch all movies on the platform since there are millions of movies in the catalog. Hence it's essential that the movies recommended to him/her are of interest to the user.

Existing recommendation algorithms are intrusive in nature. They typically collect vast amounts of sensitive information about user behavior patterns such as movies watched, items purchased etc. Such information collected by the recommendation provider poses huge privacy risks. Moreover, users will be hesitant to provide feedback information to the recommender system out of privacy concerns and this in turn will affect the quality of their recommendations. Also, existing regulations such as the General Data Protection Regulation (GDPR) [VB17] in Europe and California Consumer Privacy Act (CCPA) [Tor18] mandate strong privacy protections for users on various web platforms. Regulations are useful and required but we believe innovative technological advancements are required to advance the status quo.

Current recommender systems deployed on the web are very effective in personalization but they do so at the expense of user privacy. Due to the centralized nature of recommender systems, they become a vulnerable target for unintended consequences such as data breaches, manipulation etc. Numerous studies [GKB14] [TWW20] [LR04] [OHS05] [FGL20] have shown the risks involved in targeted attacks against recommender systems that have centralized storage of user data. Hence, what's currently required is a framework that empowers users to decide their own privacy policy independent of the recommender system provider.

More specifically, we believe:

1. All data generated by the user must be stored locally and never shared with anyone.

2. User's should be able to decide their own privacy levels using local differential privacy.

3. Users ought to be able to quantify the loss of privacy from the recommendations generated by the server.

5.2 Background

5.2.1 Collaborative Filtering

Collaborative Filtering is a very popular technique used in personalization engines across the web. It is based on the principle of homophily, i.e. similar users tend to like similar items. We model user feedback in the form of ratings which represent a real value on the likert scale. Generally, a higher rating value implies greater interestpreference for an item. We represent the rating given by user u on a item i via r_{ui}. Without loss of generality, ratings can also model implicit feedback that represents whether a user has interacted with an item or not. Implicit feedback is easier to collect since it does not require much cognitive effort on the user's behalf but has less information gain compared to explicit ratings.

The traditional view of recommender systems is that of a sparse matrix, whereby a rows represents an individual user and a column represents an item. The entry in the matrix corresponds to the rating provided by the (user, item) combination. Matrix factorization [BGM15] has proven to be an effective and scalable technique in collaborative filtering. The goal of matrix factorization is to convert the sparse rating matrix into dense matrices, that embed users and items into points in a latent space. Each user, item can represented using a dense vector, namely: p_u & q_i respectively. Every useritem also has a bias for it‘s rating value which can be modeling using b_u & b_i and μ captures the global mean of ratings. The predicted rating for user u on item i, is estimated as follows:

$$\hat{r_{ui}} = \mu + b_u + b_i + q_i^T p_u \tag{5.1}$$

The overall loss function over all ratings in the training dataset is as follows:

$$\min \sum_{r_{ui} \in \Omega} (r_{ui} - \hat{r_{ui}})^2 + \lambda (b_i^2 + b_u^2 + \| q_i \|_2^2 + \| p_u \|_2^2) \tag{5.2}$$

with λ being the regularization parameter. The partial derivatives with respect to each of our parameters are as follows:

$$\begin{aligned} b_u &\leftarrow b_u + \delta(e_{ui} - \lambda b_u) \\ b_i &\leftarrow b_i + \delta(e_{ui} - \lambda b_i) \\ p_u &\leftarrow p_u + \delta(e_{ui} q_i - \lambda p_u) \\ q_i &\leftarrow q_i + \delta(e_{ui} p_u - \lambda q_i) \end{aligned} \tag{5.3}$$

where e_{ui} represents the difference between predicted and actual rating.

5.2.2 Federated Learning

In traditional machine learning, the data resides solely on a central server that performs all the necessary computation. Federated learning [LST20] enables distributed machine learning by letting users keep all the data local on their devices. In the federated learning setting, each client is responsible for gradient computation using their local data and the central server's primary role is that of aggregation. This decentralized framework of learning has gained a lot of popularity due to it's applicability to the real world. In the real world, user devices are heterogeneous in nature and the asynchronous nature enables users to join/leave the system anytime. Most importantly, federated learning enables users to keep data private by storing it locally on their device and not sharing it with anyone, including the central server.

Flanagan et al. [FOG20] implement the matrix factorization in the federated learning setting. Their work takes into account additional sources of information in the recommendation domain such as user and movie features. They claim that by jointly factorizing the matrices into a joint low-dimensional representation, they are able to achieve better quality recommendations.

The current challenges of federated learning are mainly two-fold: communication and client-side computation cost. During each round, each client is required to provide gradient updates to the server which incurs a communication cost. Since the user devices are heterogeneous in nature, the communication medium could be wired connections such as ethernet, cable etc. to wireless mediums such as WiFi, 5G etc. The other major cost incurred by clients is that of computation complexity. Since user devices are mostly light-weight, the computation load has to lower since there are constraints such as energy consumption.

5.2.3 Local Differential Privacy

Differential privacy [DR14] [ACG16] [MPR09] is considered to be a gold-standard in the realm of privacy-preserving methods. Differential privacy provides theoretical guarantees and quantifies the information loss in the form of a privacy budget.

Definition: A randomized algorithm $\mathscr{M}$ with domain $\mathscr{N}^{|\mathscr{X}|}$ is (ε, δ)-differentially private if for all $\mathscr{S} \subseteq Range(\mathscr{M})$ and for all $x, y \in \mathscr{N}^{|\mathscr{X}|}$ such that $||x-y||_1 \leq 1$:

$$Pr[\mathscr{M}(x) \in \mathscr{S}] \leq exp(\varepsilon) Pr[\mathscr{M}(y) \in \mathscr{S}] + \delta \tag{5.4}$$

where the probability space is over the coin flips of mechanism $\mathscr{M}$. Also, if $\delta = 0$, we say that $\mathscr{M}$ is ε-differentially private.

The conventional differential privacy setting consists of a central server where all the data resides and the goal is to protect user privacy from external clients. On the other hand, local differential privacy deals with the setting whereby privacy is being offered to users from the central server itself. The goal being that user data can reside locally on their devices and central server simply acts as an aggregator.

The main parameter controlling the privacy in differential privacy is ε, known as the privacy budget. Lower epsilon values implies better privacy, lower utility and vice-versa holds true.

5.3 Motivation

Existing recommender systems rely on extensive information based on user input and interactions on their platforms. We argue that such information not only captures the macro level population trends but can also paint a detailed picture on an individual user level. Such data collection practices can seem benign but over time it can used to accurately predict sensitive information including protected classes such as: gender, race, religion, political preference etc.

5.4 Threat Model

Most recommender systems consist of a centralized server where all the ratings information across various users is stored. Such existing approaches are not privacy friendly since the assumption is that the centralized server is trustworthy which is often not the case.

Our federated approach ensures that all ratings information stays on the respective user's device and the central server only serves as an aggregator and generates the recommendations. The central premise of our approach is that the raw rating information never leaves the user device.

We plan to use local differential privacy wherein which allows satisfies all our requirements.

5.4.1 Role of Central Server

The central server plays an important role in our framework. Complete decentralized recommender systems are not scalable with the number of users & items. Decentralized recommender systems have been shown [SFR06] to suffer from a variety of issues such as: bias or sabotage from malicious users, poor quality of recommendations etc.

The central server in our setting has a vested interest to provide high quality recommendations to users meanwhile preserving individual user privacy. The central server is assumed to have sufficient computational resources that can scale to millions of users and items. Moreover, the server is assumed to be "curious", i.e. it tries to learn as much information as possible about a user in order to provide better recommendations.

5.5 Practical Implications of Differential Privacy

The intuition behind the definition of differential privacy is that just by viewing the output of an algorithm it should be hard for an attacker to know if a particular user's information was included in the computation or not. Consider a medical database **D** and an algorithm $\hat{A}$ that is able to run analytical queries on **D**. Suppose, that the result of $\hat{A}$ on **D** is *Result I*. Now, if we add in the medical information for a user Joe to the medical database **D** and pass it through our algorithm $\hat{A}$, say we get the result *Result II*. The goal of Differential Privacy that an attacker should be easily able to

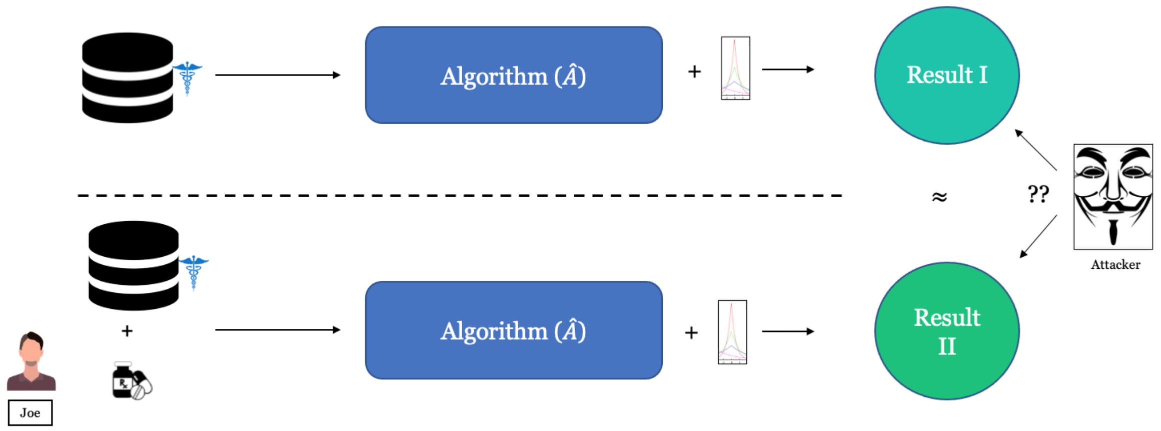


Figure 5.1: Motivation of Differential Privacy

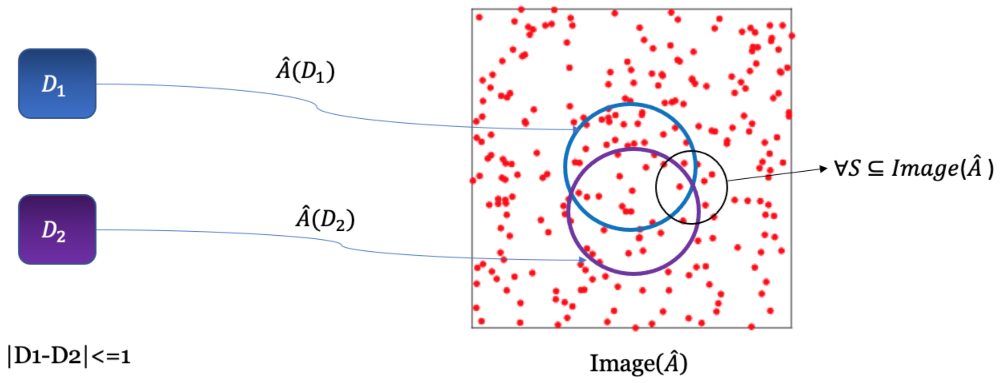


Figure 5.2: Stochasticity of a differentially private algorithm

distinguish between the 2 inputs to $\hat{A}$ simply by looking at the two outputs: *Result I* and *Result II*. By definition, differential privacy makes no assumptions about the external knowledge that is accessible to an attacker.

The stochastic nature of the DP algorithm $\hat{A}$, ensures that the output is not the same for the same input. Stochasticity plays an important role in making an algorithm differentially private.

Given 2 datasets D_1 and D_2 that differ in at most 1 element, we would like the outputs to be similar to each other in order to make $\hat{A}$ a strong differentially private algorithm.

The stochastic nature of the algorithm ensures that for the same input D_1 the output results in the set indicated by the blue arrow as shown in Figure 5.2. Similarly, the input D_2 can result in an output set pointed by the purple border in the image of the algorithm $\hat{A}$. In order for an algorithm to have strong differential private guarantees, the overlap of the two output sets has to be high i.e. given a subset S of image($\hat{A}$), the ratio of the probability that they output belongs to S is bounded by e^{ε}. ε is the parameter that quantifies the privacy in a differentially private algorithm. Lower ε value indicates a higher overlap of the two output sets, resulting in higher privacy because it becomes harder for the attacker to distinguish between the inputs. On the other hand, a higher ε value indicates a lower overlap between the two output sets resulting in lower privacy.

- Quantification of privacy loss:

  Differential Privacy provides rigorous theoretical bounds on the privacy lost/gained in during a transformation. Prior to Differential Privacy, techniques such as anonymization have been shown to fail [Ohm09] in safeguarding user privacy due to linkage attacks. For example, Sweeney et al. [Swe02] have shown that k-anonymity [ED08] is not effective in protecting personal information. Sweeney et al. have been able to "deanonymize" the personal record of the Massachusetts Governor by cross-referencing anonymous public health and voter records. Such attacks are known as linkage attacks and are based on connections to external sources of information. Linkage attacks are proven to be successful in the recommender systems domain with the Netflix challenge [NS06] and the genomic data [SAW13].

- Composition: Differentially private mechanisms have a nice property that their compositions are closed, i.e. the resulting mechanism is also differentially private.

5.6 TensorFlow Privacy

Tensorflow Privacy is an open-source Python library that includes the implementations of optimizer used for training machine learning models in a differentially private manner. The library is equipped with various analysis tools required for computing the privacy guarantees provided by the differentially private optimizers.

As mentioned on the TensorFlow Privacy Github page, the main difference between traditional optimizers for the Stochastic Gradient Descent and the Differentially Private versions are:

- l2_norm_clip: The cumulative gradient across all network parameters from each microbatch will be clipped so that its L2 norm is at most this value.

  item noise_multiplier: This governs the amount of noise added during training. Generally, more noise results in better privacy and lower utility.

The common parameters across the 2 optimizers are:

- learning_rate: This value determines the impact of each update. In practice, exponential decay of learning rate is used for the procedure to converge.
- num_microbatches: This parameter determines the number of examples consumed in each gradient update step. Generally, increasing the number of microbatches will improve the utility but slow down the training time since there are more number of update iterations to be performed per batch.

The library also provides tools to perform privacy budgeting i.e. computes the ε value for various settings. Mironov et al. [MTZ19] provide a closed-form bound for the epsilon value when Gaussian noise is added to gradients in machine learning applications.

5.6.1 Individual-level user privacy

The existing one-size-fits-all approach to privacy in recommender systems does not work. Our framework aims to tackle this issue by providing control to the users over their individual privacy policy. Using local differential privacy, we are able to provide more fine-grained control to the user whereby he/she can decide their individual privacy policy.

5.6.2 Appropriate ε value

The advantage of Differential Privacy is that it is able to mathematically quantify the privacy loss/-gain. As mentioned on the TensorFlow Privacy Github page, differential privacy can be expressed using the following 2 values:

- Epsilon

  This value provides a ceiling on how much the probability of particular output can increase by including (or removing) a single training example. The range of ε is $[0,\infty)$. A lower epsilon value implies higher privacy guarantee and vice-versa.

- Delta

  Delta bounds the probability of an arbitrary change in model behavior. It is usually set to a very small number so as to avoid compromising utility. Rule of thumb is to set it to be less than the inverse of the training data size.

5.7 Proofs

5.7.1 Proof I: Sequential Composition of Users across Datasets

Claim: The final ε result of composition across multiple differentially private algorithms, results in their additive value of $epsilon$ value per individual user.

Consider the following scenario: Company A and Company B have implemented our framework of federated local differential privacy based recommendation. Say, both the companies have a subset of users in common and that Company A has decided to acquire Company B. They have decided to integrate the user and item vectors across the two companies. According to Kairouz et al. [KOV15], the composition theorem applies to the determine the final ε value for a user. If a user U, had a privacy budget of ε_1 in Company A and a privacy budget of ε_2 in Company B, then the new combined privacy budget for user U would be $\varepsilon_1 + \varepsilon_2$. This implies, that new combined company would have access to more information about user U, but it is bounded by the epsilon value of $\varepsilon_1 + \varepsilon_2$.

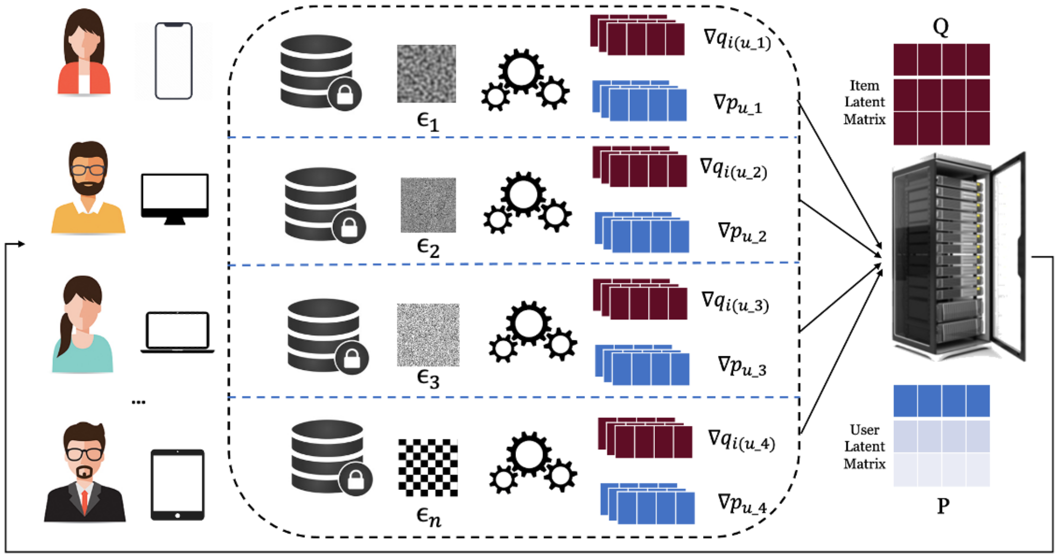


Figure 5.3: Our Proposed Framework

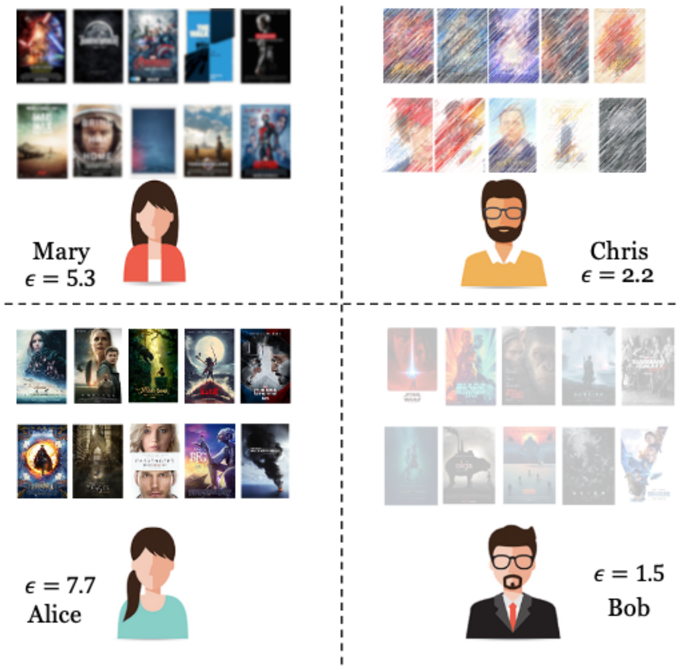


Figure 5.4: Individual Privacy Controls

5.8 Framework

5.8.1 Individual Privacy Controls

Our framework provides the option for individual users to decide their independent privacy level. As illustrated in the Figure 5.4, each user can decide their privacy level by selecting the appropriate ε value. A lower epsilon value indicates higher privacy level and vice-versa, a higher epsilon value indicates lower privacy constraints. Based on the example in Figure 5.4, Bob has the lowest $\varepsilon = 1.5$ value indicating a conservative behavior by selecting a high privacy threshold, which is why the the Central Recommender Server (CRS) will not be able to accurately infer the items consumed by Bob. On the other hand, Alice has the highest $\varepsilon = 7.7$ value among the 4 users, indicating a lower privacy threshold.

There is an inherent tradeoff between epsilon and the accuracy of the recommendations. The lower ε value indicates a higher privacy threshold,

5.8.2 Stochastic Gradient Descent

We are interested in minimizing the following loss function:

$$\min \sum_{r_{ui} \in \Omega} (r_{ui} - \hat{r_{ui}})^2 + \lambda (b_i^2 + b_u^2 + \parallel q_i \parallel_2^2 + \parallel p_u \parallel_2^2) \tag{5.5}$$

5.8.3 Client Side Computation

On the client side, each individual user performs the below updates using their local ratings information:

$$\begin{aligned} b_u &\leftarrow b_u + \delta(e_{ui} - \lambda b_u) \\ b_i &\leftarrow b_i + \delta(e_{ui} - \lambda b_i) \\ p_u &\leftarrow p_u + \delta(e_{ui} q_i - \lambda p_u) \\ q_i &\leftarrow q_i + \delta(e_{ui} p_u - \lambda q_i) \end{aligned} \tag{5.6}$$

Each user/client is responsible for computation over their private rating information. So the steps involved are:

- Determine the noise multiplier

  Our framework enables users to determine their own epsilon value, rough measure of differential privacy. The ε value is determined based factors such as: number of examples in training data, batch size, noise multiplier, number of epochs and the delta value used in training.

  - Number of examples in training data: For a particular user, the number of training examples corresponds to the number of ratings they have provided. This value changes over time but for the purposes of training, we can assume this quantity to be fixed.
  - Batch Size: Batch size controls the number of training samples processed prior to updating the model parameters. This value can be selected by our framework as a standard depending on the number of training examples by a user.
  - Noise Multiplier: Noise multiplier determines the amount of noise to be added when performing gradient computation. TensorFlow provides a mechanism to compute the ε value given all other parameter values. We have devised a binary search based method to compute the noise multiplier for a required setting.
  - Number of epochs: The number of epochs affects the final ε value and our case we can set it to be a fixed value across the users.

- Perform update rules The item latent vectors are retrieved from the server since they are shared across users. The user latent vector is initialized to a random vector. The update rules are performed using the equations above using the differentially private optimizer. It is important to note that this optimizer is user-specific since it determines the final ε value.

- Return the user and item embeddings to the server Once the user and item latent vectors are updated, they are both returned to the server.

The item latent vectors are shared across all users and synchronously updated so as to ensure they converge.

The user latent vectors can be shared to the server since they are guaranteed to be deferentially private. Moreover, after the training process, user latent vectors are required to perform ranking over items and generate recommendations.

5.8.4 Server side Computation

Although the client-side performs the crucial gradient computation for the user and item latent vectors, the server plays a crucial role in coordination across users. The assumptions we make are the following:

- Large number of items: We assume that the set of items is large, making it out of reach for a single user/device to store all the required information.
- Resource intensive central server: We assume that the central server has sufficient computational resources to perform the heavy-lifting computation.
- User device can perform light-weight computation: Since we are taking a federated learning approach, the user device can perform the required differentially private stochastic gradient computation locally and also store the ratings provided by the user locally on device.

We believe the above assumptions are very reasonable and mimic the real-world to a large extent.

5.8.4.1 Recommendation List Generation

An important step once the training process has been completed is to generate the recommendations. The server which has access to the user and item latent vectors can rank the items based on the predicted rating value and serve them to the respective user.

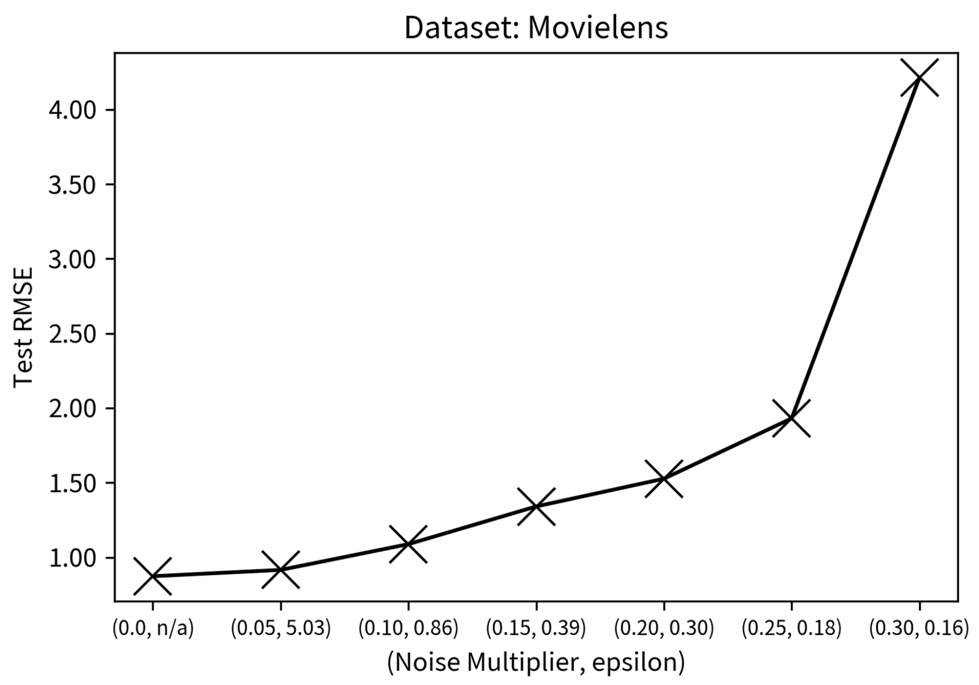


Figure 5.5: Epsilon vs RMSE on Movielens

5.9 Results

We run experiments on two popular datasets, namely: Movielens and Amazon video ratings.

In Figure 5.5, we show the tradeoff between the ε value and the Test Root Mean Square Error (RMSE), which measures the average root value of the squared difference between the actual rating and predicted rating provided by the user. Lower RMSE indicates better recommendation accuracy.

Based on the result plot, we observe that the RMSE value grows as expected when we increase the noise multiplier. It's interesting to note that the RMSE value grows exponentially as we increase the noise multiplier or lower ε value, which indicates higher privacy threshold. This graph also enables an user to determine his/her appropriate ε value. The user can determine the drop in accuracy they are willing to accept based on the corresponding ε value.

We also plot the Test RMSE during various epochs of training for different noise values as shown in Figure 5.6. As we can expect, the test RMSE converges across all the noise values, but

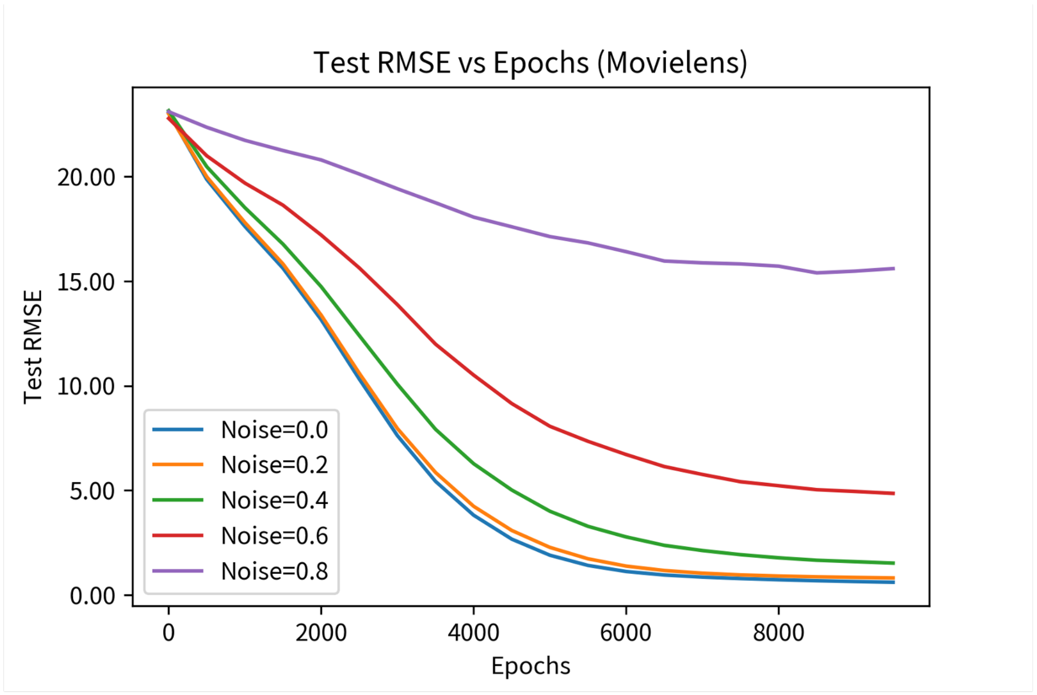


Figure 5.6: Test RMSE over epochs in Movielens

the test RMSE and noise value are proportional to each other.

5.10 Related Work

5.10.1 Privacy Preserving Recommendation

Privacy preserving recommender systems has garnered interest in the research community. Broadly, the two main parties involved are: the central recommender system server and its users. Broadly, various approaches can be categorized based on the whether privacy is being protected from the central server or other users in the system.

5.10.1.1 Privacy from central server

Minto et al. [MHH21] propose a similar framework to our approach where they employ local differential privacy in a federated setting to protect user rating information from the central server. While we appreciate their spirit and direction, their approach is not feasible in the practical setting. Firstly, their parameter k which controls the number of ε-LDP gradient updates by each user has to large in order to provide decent quality recommendations. Moreover, with a large value of k,

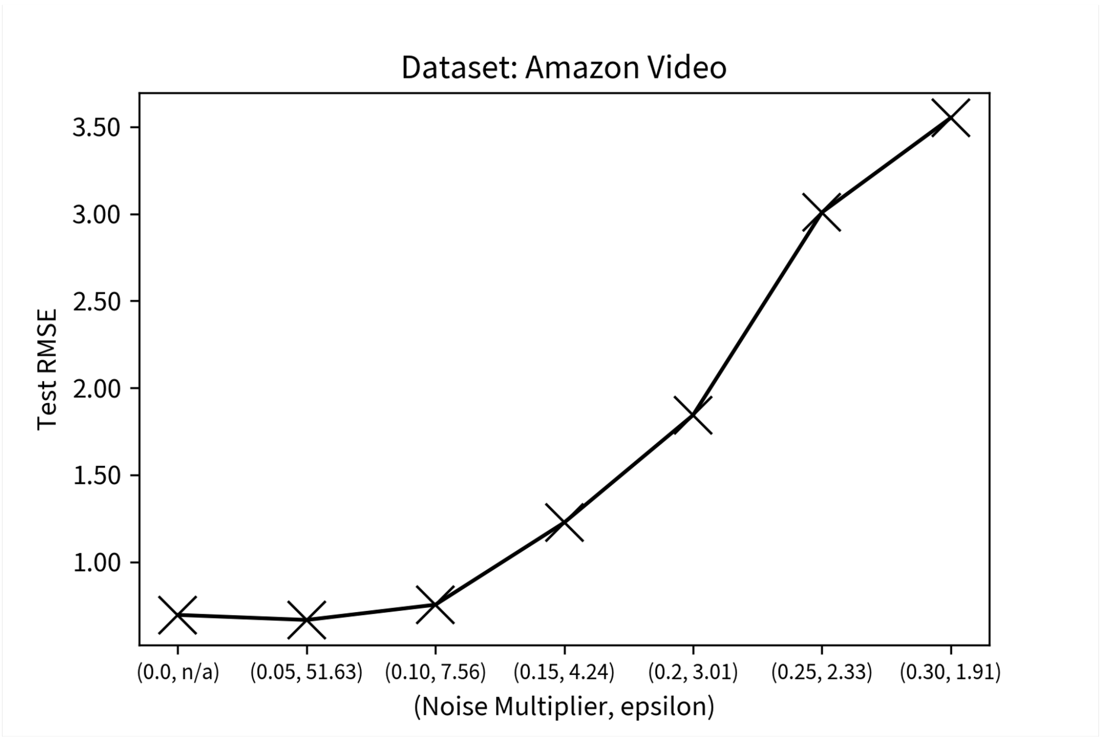


Figure 5.7: Epsilon vs RMSE on Amazon Video

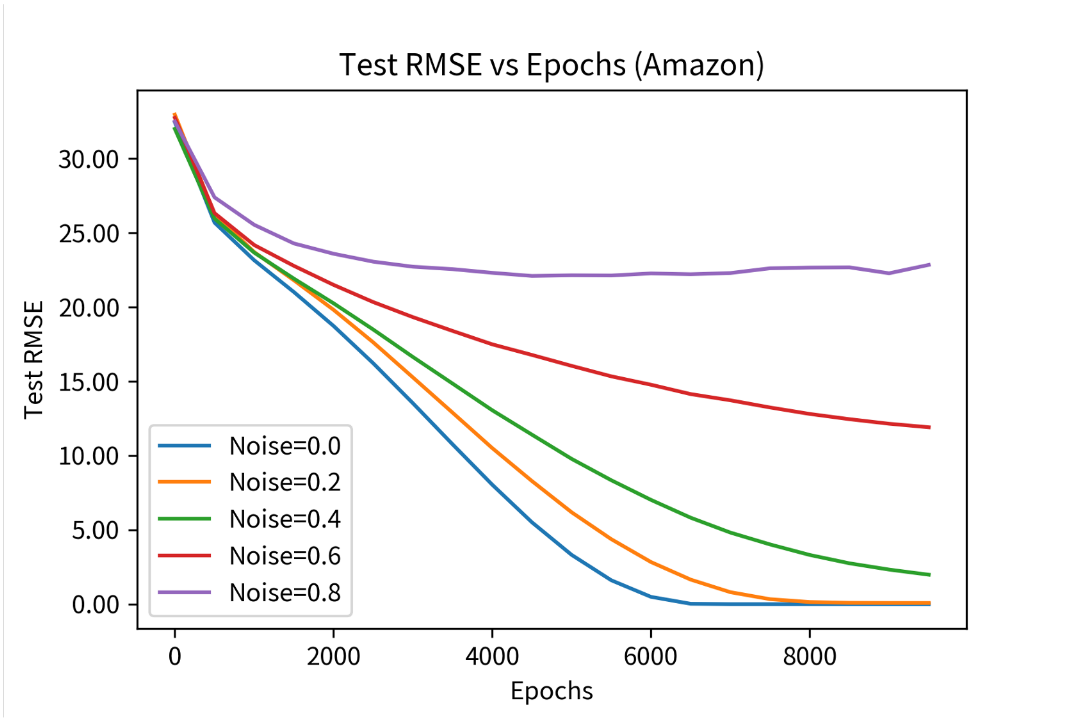


Figure 5.8: Test RMSE over epochs in Amazon

the overall privacy guarantee becomes (k*ε) differentially private, which can substantially large if k is larget than 100. The authors also introduce a proxy network that lies in between the user devices and the central server. They mention that the goal of the proxy network is to "reduce the fingerprinting surface by anonymizing and shuffling the reports before forwarding them to the recommender". In practice, the central server would most likely have to implement this proxy server resulting in a conflict of interest. This will render the proxy network useless since there is no guarantee that the central server will not correlate the gradient updates to individual users.

Another drawback of Minto et al.'s framework is that they do not provide individual users with control over their privacy. Whereas in our approach, we allow each user to decide his/her own ε value for differential privacy. This empowers users to determine their own privacy policy.

In the real world setting, there are millions of items in the recommender system domain. Therefore, on-device recommendation would not feasible since the user would have to download all the item latent vectors and then perform a ranking over them in order to obtain a top-n recommendation. This is not feasible due to the communication and computation constraints on the user devices. Hence, in our framework, even though the user latent vectors reside on the server, they are guaranteed to be ε_k differentially private for user k. The server can then perform the computationally intensive dot product to obtain the top-n item recommendations for the user.

5.10.1.2 Privacy from other users

Privacy preserving recommender systems that aims to protect users from other users can be broadly classified into two groups: distributed and cryptographic approaches.

Berkovsky et al [BEK07b] propose a distributed collaborative filtering based approach whereby there is no centralized storage of user feedback data. The authors propose a peer-to-peer aggregation strategy with data obfuscation involved. Similarly, Shokri et al. [SPT09b] alter the distributed collaborative filtering approach by performing user aggregation instead of perturbation. Such distributed techniques are subject to attacks from malicious users and not scalable for the real-world setting.

Cryptographic protocols such as homomorphic encryption, allows the server to perform computation on ciphered data without revealing the underlying content. Erkin et al. [EBV11] propose such a collaborative filtering technique using homomorphic encryption and secure multiparty computation techniques. The main drawback of such techniques is that they are not efficient and do not scale to millions of users and items.

5.10.2 Differential Privacy

5.10.2.1 Central Differential Privacy

McSherry et al. [MM09b] proposed the initial approach to applying differential privacy to the recommender systems domain. They consider the centralized approach whereby the server which has access to all the ratings computes the latent factors in a differentially private manner, so as to protect privacy from external agents that might interact with the central server. Their paper was the first to demonstrate the trade-off between accuracy and various privacy guarantees. Their approach also relies on the perturbation of the user information prior to sending it to the central server. The amount of perturbation or noise added determines the privacy protection offered to a user.

Another direction undertaken by several researchers [AS11] [Can02a] [NIW13] include the use of cryptography techniques such as homomorphic encrption and Yao's garbled circuits. These approaches are mostly theoretical and not practically feasible since they are not able to scale in terms of their computational complexity.

5.10.2.2 Local Differential Privacy

Local Differential Privacy has gained popularity in recent years due to benefits such as: eliminating need for central data repository and theoretical privacy guarantees. Moreover, the local differential privacy concept lends itself very well to federated learning whereby the data remains locally on the user's device and training is performed through aggregated gradient updates.

The tech industry has been quick to adapt local differential privacy for their product offerings.

For example, Google proposed the use of RAPPOR [EPK14], a mechanism for collecting end-user statistics such as most popular site etc. using randomized response techniques. RAPPOR employs the use of Bloom filters to provide a configurable framework that can provide strong

ε-differential privacy guarantees.

In 2016, Apple Inc. published a whitepaper and accompanying patent describing the use of local differential privacy for tasks such as detecting the most popular emoji across users. They employ the Hadamard Count Mean-based sketch technique which injects noise and performs sampling to preserve user privacy.

Microsoft has proposed [DKY17] a local differential privacy based solution that aims to collect statistics over long period of time. They focus on the tasks of mean & histogram estimation and claim that it has been deployed to millions of devices to collect telemetry data.

CHAPTER 6

Conclusion and Future Directions

Recommender systems are ubiquitous across the web. They are used to drive engagement across various domains such as movies, music, products etc. Current recommender systems require all user feedback information to stored on the central recommendation server. The central information server collects information about the items interacted across users in the system. This information trove can be used to accurately determine the preferences of an individual user on sensitive topics such as health, politics etc. This leads to serious privacy concerns. In this thesis, we propose three paradigms that enable privacy-preserving recommender systems. We incrementally show how users can share less data with the central recommender system and yet receive high quality recommendations.

In the chapter Bayesian prior learning for next item recommendation, we demonstrate how the central recommender system can anonymize the sequences of items consumed by a user and only rely on the frequency of n-grams in the training dataset to perform recommendations in an "in-cognito" fashion. Such a recommender system simply uses the context information to predict the next of items a user is likely to interact with. The assumption made is that users trust the central recommender server with de-linking the sequential information with unique user identifier information. In our second framework, Selective Privacy Collaborative Filtering, we relax this assumption by allowing users to decide which data they would like to share with the central recommender system. In essence, we are allowing each user to determine their individual privacy policy. Our framework initially builds a public model based on the ratings information shared publicly by all users. The next step involves each user fine-tuning their individual latent vector based on their private ratings on-device. This final fine-tuned model is then stored locally and never shared. In our experiments, we simulate various user behaviours in determining the ratio of items stored as public/private. In our third framework, we provide the option of storing the entire data locally and utilize the notion of Differential Privacy provide theoretical privacy guarantees to the user. We employ a federated learning approach whereby we allow each user to decide a privacy budget and the gradient updates are guaranteed to align this budget through gradient clipping and addition of noise. We demonstrate the tradeoffs between noise and utility in terms of RMSE on two real world recommender systems datasets.

For future work in privacy preserving recommender systems, it can be categorized into three areas, namely: efficiency (computational and communication), robustness and group privacy.

6.1 Computational and Communication Efficiency

Since the edge devices used by users might be constrained by energy requirements, it becomes imperative to be efficient in terms of the computational and communication cost. Batch processing techniques can be employed to reduce the number of requests made to the central recommender system in an asynchronous fashion.

6.2 Robustness

Federated learning for recommender systems needs to be robust to malicious actors since it becomes difficult for the server to distinguish between gradients with noise vs gradients targeted for a specific purpose. Hence, robustness ensures that the utility of the framework remains high meanwhile protecting user privacy.

6.3 Group Privacy

The third future research direction would be in the area of group privacy. Currently, privacy is being analyzed in terms of an individual but more work is required to understand group dynamics if recommendations are provided collectively to a set of users.

This thesis aims to motivate the importance of privacy preserving recommender systems and proposes three paradigms that empower an individual to determine their personalized privacy policy. We are confident that future research will build upon our ideas in this important and exciting area of privacy preserving recommendations.

References

[ACD11] Artin Armagan, Merlise Clyde, and David B Dunson. "Generalized beta mixtures of Gaussians." In *Advances in neural information processing systems*, pp. 523–531, 2011.

[ACG16] 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*, pp. 308–318, 2016.

[Adv87] Stephen Advokat. "Publication of Bork's video rentals raises privacy issue." *Chicago Tribune*, 1987.

[AS11] Frederik Armknecht and Thorsten Strufe. "An efficient distributed privacy-preserving recommendation system." In *2011 The 10th IFIP Annual Mediterranean Ad Hoc Networking Workshop*, pp. 65–70. IEEE, 2011.

[AT05] Gediminas Adomavicius and Alexander Tuzhilin. "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions." *IEEE transactions on knowledge and data engineering*, **17**(6):734–749, 2005.

[AV07] David Arthur and Sergei Vassilvitskii. "K-means++: The Advantages of Careful Seeding." In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pp. 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[BEK07a] Shlomo Berkovsky, Yaniv Eytani, Tsvi Kuflik, and Francesco Ricci. "Enhancing Privacy and Preserving Accuracy of a Distributed Collaborative Filtering." In *Proceedings of the 2007 ACM Conference on Recommender Systems*, RecSys '07, pp. 9–16. ACM, 2007.

[BEK07b] Shlomo Berkovsky, Yaniv Eytani, Tsvi Kuflik, and Francesco Ricci. "Enhancing privacy and preserving accuracy of a distributed collaborative filtering." In *Proceedings of the 2007 ACM conference on Recommender systems*, pp. 9–16, 2007.

[BGM15] Dheeraj Bokde, Sheetal Girase, and Debajyoti Mukhopadhyay. "Matrix factorization model in collaborative filtering algorithms: A survey." *Procedia Computer Science*, **49**:136–146, 2015.

[BIK17] 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." Cryptology ePrint Archive, Report 2017/281, 2017. `https://eprint.iacr.org/2017/281`.

[Blo70] Burton H. Bloom. "Space/Time Trade-offs in Hash Coding with Allowable Errors." *Commun. ACM*, **13**(7):422–426, July 1970.

[BR03] Ferenc Bodon and Lajos Rónyai. "Trie: an alternative data structure for data mining algorithms." *Mathematical and Computer Modelling*, **38**(7-9):739–751, 2003.

[Can02a] John Canny. "Collaborative filtering with privacy." In *Proceedings 2002 IEEE Symposium on Security and Privacy*, pp. 45–57. IEEE, 2002.

[Can02b] 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, pp. 238–245. ACM, 2002.

[CLS18] Ching-Wei Chen, Paul Lamere, Markus Schedl, and Hamed Zamani. "Recsys challenge 2018: automatic music playlist continuation." In *Proceedings of the 12th ACM Conference on Recommender Systems*, pp. 527–528, 2018.

[CPW12] Richard Chow, Manas A. Pathak, and Cong Wang. "A Practical System for Privacy-Preserving Collaborative Filtering." In *12th IEEE International Conference on Data Mining Workshops, ICDM Workshops, Brussels, Belgium, December 10, 2012*, pp. 547–554, 2012.

[CR95] Yves Chauvin and David E Rumelhart. *Backpropagation: theory, architectures, and applications*. Psychology press, 1995.

[CZL19] Qiwei Chen, Huan Zhao, Wei Li, Pipei Huang, and Wenwu Ou. "Behavior sequence transformer for e-commerce recommendation in alibaba." In *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*, pp. 1–4, 2019.

[DCL18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805*, 2018.

[DDY19] Manoj Reddy Dareddy, Mahashweta Das, and Hao Yang. "motif2vec: Motif aware node representation learning for heterogeneous networks." In *2019 IEEE International Conference on Big Data (Big Data)*, pp. 1052–1059. IEEE, 2019.

[DKY17] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. "Collecting telemetry data privately." *arXiv preprint arXiv:1712.01524*, 2017.

[DR14] Cynthia Dwork, Aaron Roth, et al. "The algorithmic foundations of differential privacy." *Found. Trends Theor. Comput. Sci.*, **9**(3-4):211–407, 2014.

[DUM17] Ariyam Das, Ishan Upadhyaya, Xiangrui Meng, and Ameet Talwalkar. "Collaborative Filtering As a Case-Study for Model Parallelism on Bulk Synchronous Systems." In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pp. 969–977. ACM, 2017.

[EBV11] Zekeriya Erkin, Michael Beye, Thijs Veugen, and Reginald L Lagendijk. "Efficiently computing private recommendations." In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5864–5867. IEEE, 2011.

[ED08] Khaled El Emam and Fida Kamal Dankar. "Protecting privacy using k-anonymity." *Journal of the American Medical Informatics Association*, **15**(5):627–637, 2008.

[EPK14] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. "Rappor: Randomized aggregatable privacy-preserving ordinal response." In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pp. 1054–1067, 2014.

[FGL20] Minghong Fang, Neil Zhenqiang Gong, and Jia Liu. "Influence function based data poisoning attacks to top-n recommender systems." In *Proceedings of The Web Conference 2020*, pp. 3019–3025, 2020.

[FOG20] Adrian Flanagan, Were Oyomno, Alexander Grigorievskiy, Kuan Eeik Tan, Suleiman A Khan, and Muhammad Ammad-Ud-Din. "Federated multi-view matrix factorization for personalized recommendations." *arXiv preprint arXiv:2004.04256*, 2020.

[GKB14] Ihsan Gunes, Cihan Kaleli, Alper Bilge, and Huseyin Polat. "Shilling attacks against recommender systems: a comprehensive survey." *Artificial Intelligence Review*, **42**(4):767–799, 2014.

[GLM16] Rong Ge, Jason D. Lee, and Tengyu Ma. "Matrix Completion Has No Spurious Local Minimum." In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 2981–2989. Curran Associates Inc., 2016.

[GR06] Oleg Grodzevich and Oleksandr Romanko. "Normalization and other topics in multi-objective optimization." In *Proceedings of the Fields-MITACS Industrial Problems Workshop*, 08 2006.

[GW99] Claudio Gentile and Manfred KK Warmuth. "Linear hinge loss and average margin." In *Advances in neural information processing systems*, pp. 225–231, 1999.

[HK15] F. Maxwell Harper and Joseph A. Konstan. "The MovieLens Datasets: History and Context." *ACM Trans. Interact. Intell. Syst.*, **5**(4):19:1–19:19, December 2015.

[JVS16] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. "Exploring the limits of language modeling." *arXiv preprint arXiv:1602.02410*, 2016.

[KBB18] Domokos M Kelen, Dániel Berecz, Ferenc Béres, and András A Benczúr. "Efficient K-NN for playlist continuation." In *Proceedings of the ACM Recommender Systems Challenge 2018*, pp. 1–4. 2018.

[KBV09] Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix Factorization Techniques for Recommender Systems." *Computer*, **42**(8):30–37, August 2009.

[KM16] Efthalia Karydi and Konstantinos Margaritis. "Parallel and Distributed Collaborative Filtering: A Survey." *ACM Comput. Surv.*, **49**(2):37:1–37:41, August 2016.

[KMN02] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. "An Efficient k-Means Clustering Algorithm: Analysis and Implementation." *IEEE Trans. Pattern Anal. Mach. Intell.*, **24**(7):881–892, July 2002.

[KMR16] Jakub Konećný, H. Brendan McMahan, Daniel Ramage, and Peter Richtarik. "Federated Optimization: Distributed Machine Learning for On-Device Intelligence.", 2016.

[KOV15] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. "The composition theorem for differential privacy." In *International conference on machine learning*, pp. 1376–1385. PMLR, 2015.

[KP15] Janis Keuper and Franz-Josef Pfreundt. "Asynchronous Parallel Stochastic Gradient Descent: A Numeric Core for Scalable Distributed Machine Learning Algorithms." In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, MLHPC '15, pp. 1:1–1:11. ACM, 2015.

[LCG17] Yuan Liu, Usman Shittu Chitawa, Guibing Guo, Xingwei Wang, Zhenhua Tan, and Shuang Wang. "A Reputation Model for Aggregating Ratings Based on Beta Distribution Function." In *Proceedings of the 2nd International Conference on Crowd Science and Engineering*, ICCSE'17, pp. 77–81. ACM, 2017.

[Lin07] Chih-Jen Lin. "Projected Gradient Methods for Nonnegative Matrix Factorization." *Neural Comput.*, **19**(10):2756–2779, October 2007.

[LLL16] Yanchi Liu, Chuanren Liu, Bin Liu, Meng Qu, and Hui Xiong. "Unified Point-of-Interest Recommendation with Temporal Interval Assessment." In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 1015–1024, New York, NY, USA, 2016. ACM.

[LLS16] Mu Li, Ziqi Liu, Alexander J. Smola, and Yu-Xiang Wang. "DiFacto: Distributed Factorization Machines." In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, WSDM '16, pp. 377–386. ACM, 2016.

[LR04] Shyong K Lam and John Riedl. "Shilling recommender systems for fun and profit." In *Proceedings of the 13th international conference on World Wide Web*, pp. 393–402, 2004.

[LS00] Daniel D. Lee and H. Sebastian Seung. “Algorithms for Non-negative Matrix Factorization.” In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS’00, pp. 535–541. MIT Press, 2000.

[LSJ16] Jason D. Lee, Max Simchowitz, Michael I. Jordan, and Benjamin Recht. “Gradient Descent Only Converges to Minimizers.” In Vitaly Feldman, Alexander Rakhlin, and Ohad Shamir, editors, *29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pp. 1246–1257. PMLR, 23–26 Jun 2016.

[LST20] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. “Federated learning: Challenges, methods, and future directions.” *IEEE Signal Processing Magazine*, **37**(3):50–60, 2020.

[LZL18] Zhi Li, Hongke Zhao, Qi Liu, Zhenya Huang, Tao Mei, and Enhong Chen. “Learning from history and present: Next-item recommendation via discriminatively exploiting user behaviors.” In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1734–1743, 2018.

[MB97] Hannes Marais and Krishna Bharat. “Supporting Cooperative and Personal Surfing with a Desktop Assistant.” In *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology*, UIST ’97, pp. 129–138. ACM, 1997.

[MBY16] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. “MLlib: Machine Learning in Apache Spark.” *J. Mach. Learn. Res.*, **17**(1):1235–1241, January 2016.

[McC12] Kathryn E. McCabe. “Just you and me and netflix makes three: Implications for allowing frictionless sharing of personally identifiable information under the video privacy protection act.” *J. Intell. Prop. L.*, 2012.

[MHH21] Lorenzo Minto, Moritz Haller, Hamed Haddadi, and Benjamin Livshits. “Stronger Privacy for Federated Collaborative Filtering with Implicit Feedback.” *arXiv preprint arXiv:2105.03941*, 2021.

[MM09a] Frank McSherry and Ilya Mironov. “Differentially Private Recommender Systems: Building Privacy into the Netflix Prize Contenders.” In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’09, pp. 627–636. ACM, 2009.

[MM09b] Frank McSherry and Ilya Mironov. “Differentially private recommender systems: Building privacy into the netflix prize contenders.” In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 627–636, 2009.

[MPR09] Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil Vadhan. "Computational differential privacy." In *Annual International Cryptology Conference*, pp. 126–142. Springer, 2009.

[MR16] Sonu K. Mishra and Manoj Reddy. "A Bottom-up Approach to Job Recommendation System." In *Proceedings of the Recommender Systems Challenge*, RecSys Challenge '16, pp. 4:1–4:4. ACM, 2016.

[MSC13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. "Distributed representations of words and phrases and their compositionality." In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[MTS15] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. "Image-Based Recommendations on Styles and Substitutes." In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pp. 43–52. ACM, 2015.

[MTZ19] Ilya Mironov, Kunal Talwar, and Li Zhang. "Rényi Differential Privacy of the Sampled Gaussian Mechanism.", 2019.

[NIW13] 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*, pp. 801–812, 2013.

[NLV11] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. "Can Homomorphic Encryption Be Practical?" In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, pp. 113–124, New York, NY, USA, 2011. ACM.

[NS06] Arvind Narayanan and Vitaly Shmatikov. "How to break anonymity of the netflix prize dataset." *arXiv preprint cs/0610105*, 2006.

[Ohm09] Paul Ohm. "Broken promises of privacy: Responding to the surprising failure of anonymization." *Ucla L. Rev.*, **57**:1701, 2009.

[OHS05] Michael P O'Mahony, Neil J Hurley, and Guénolé CM Silvestre. "Recommender systems: Attack types and strategies." In *AAAI*, pp. 334–339, 2005.

[OHY15] Jinoh Oh, Wook-Shin Han, Hwanjo Yu, and Xiaoqian Jiang. "Fast and Robust Parallel SGD Matrix Factorization." In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pp. 865–874. ACM, 2015.

[RFS10] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. "Factorizing personalized markov chains for next-basket recommendation." In *Proceedings of the 19th international conference on World wide web*, pp. 811–820, 2010.

[SAW13] Latanya Sweeney, Akua Abu, and Julia Winn. "Identifying participants in the personal genome project by name (a re-identification experiment)." *arXiv preprint arXiv:1304.7605*, 2013.

[SBS13] Sebastian Schelter, Christoph Boden, Martin Schenck, Alexander Alexandrov, and Volker Markl. "Distributed Matrix Factorization with Mapreduce Using a Series of Broadcast-joins." In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pp. 281–284. ACM, 2013.

[SDC19] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.", 2019.

[SFR06] K Shyong, Dan Frankowski, John Riedl, et al. "Do you trust your recommendations? An exploration of security and privacy issues in recommender systems." In *International Conference on Emerging Trends in Information and Communication Security*, pp. 14–29. Springer, 2006.

[SPT09a] 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*, RecSys '09, pp. 157–164. ACM, 2009.

[SPT09b] 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*, pp. 157–164, 2009.

[Sto13] James V Stone. *Bayes' rule: A tutorial introduction to Bayesian analysis*. Sebtel Press, 2013.

[Swe02] Latanya Sweeney. "k-anonymity: A model for protecting privacy." *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, **10**(05):557–570, 2002.

[Tor18] Lydia de la Torre. "A guide to the california consumer privacy act of 2018." *Available at SSRN 3275571*, 2018.

[TPN08] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. "Matrix Factorization and Neighbor Based Algorithms for the Netflix Prize Problem." In *Proceedings of the 2008 ACM Conference on Recommender Systems*, RecSys '08, pp. 267–274. ACM, 2008.

[TT12] Gábor Takács and Domonkos Tikk. "Alternating Least Squares for Personalized Ranking." In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pp. 83–90. ACM, 2012.

[TWW20] Jiaxi Tang, Hongyi Wen, and Ke Wang. "Revisiting adversarially learned injection attacks against recommender systems." In *Fourteenth ACM Conference on Recommender Systems*, pp. 318–327, 2020.

[VB17] Paul Voigt and Axel Von dem Bussche. "The eu general data protection regulation (gdpr)." *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, **10**:3152676, 2017.

[VSP17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[VYP17] Maksims Volkovs, Guangwei Yu, and Tomi Poutanen. "DropoutNet: Addressing Cold Start in Recommender Systems." In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pp. 4964–4973. Curran Associates, Inc., 2017.

[WHS19] Bin Wu, Xiangnan He, Zhongchuan Sun, Liang Chen, and Yangdong Ye. "ATM: An Attentive Translation Model for Next-Item Recommendation." *IEEE Transactions on Industrial Informatics*, **16**(3):1448–1459, 2019.

[WLX16] Yao Wu, Xudong Liu, Min Xie, Martin Ester, and Qing Yang. "CCCF: Improving Collaborative Filtering via Scalable User-Item Co-Clustering." In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, WSDM '16, pp. 73–82. ACM, 2016.

[Xin15] Eric P. et al. Xing. "Petuum: A New Platform for Distributed Machine Learning on Big Data." In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pp. 1335–1344. ACM, 2015.

[YA16] Lei Yang and Xavier Amatriain. "Recommending the world's knowledge: Application of recommender systems at Quora." In *Proceedings of the 10th ACM Conference on Recommender Systems*, pp. 389–389, 2016.

[ZLL17] Yu Zhu, Hao Li, Yikang Liao, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai. "What to Do Next: Modeling User Behaviors by Time-LSTM." In *IJCAI*, volume 17, pp. 3602–3608, 2017.

[ZWH08] Justin Zhan, I-Cheng Wang, Chia-Lung Hsieh, Tsan-sheng Hsu, Churn-Jung Liau, and Da-Wei Wang. "Towards Efficient Privacy-preserving Collaborative Recommender Systems." In *The 2008 IEEE International Conference on Granular Computing, GrC 2008, Hangzhou, China, 26-28 August 2008*, pp. 778–783, 2008.