Towards Recommendation Systems with Real-World Constraints

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Konstantina Christakopoulou

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Arindam Banerjee

September, 2018

Acknowledgements

First, I would like to thank my advisor Prof. Arindam Banerjee for being such an incredible mentor for me—for teaching me machine learning and how to do research in it; for guiding me from day one always with a smile while leaving me enough independence; for trusting me with such interesting and modern research topics; for supporting me during the good and the bad; for constantly pushing me for greater things (without me even realizing it); for celebrating every small victory with me; for our long discussions during important decisions; and in general for showing that he deeply cares. These past five years have exceeded my expectations. Next, I want to thank my amazing committee members Prof. George Karypis, Prof. Vipin Kumar and Prof. Gediminas Adomavicius, for agreeing to be in my committee since my written preliminary exam; for their support, positive attitude, and for their inspiring questions. Special shout-out to Prof. Karypis for helping me with advice and overall career, and for even selflessly reviewing my writing of application material for fellowships.

A very enjoyable part of my PhD has been my internships—I've been very lucky to have the most amazing mentors during all three summers outside the university. First, Katja Hoffmann and Filip Radlinski—thank you for choosing me to be your intern, and for trusting me with such a great topic. I feel that a lot of the amazing things that happened later in my PhD can be attributed to the summer 2015 that I spent with you in Cambrige, and I am very grateful for that. Katja, thank you for being so inspiring. Filip, thank you for being such a great mentor for me even after the internship—I feel lucky that you shelflessly give me your advice. Then, Adam Kalai; thank you for being such a great mentor, for your support and advice and for treating me amazingly—I am very happy for the summer we spent doing machine learning for program synthesis; it opened my eyes to new possibilities for machine learning. Next, I want to thank Ed Chi, Alex Beutel, Sagar Jain, Rui Li, for the great summer I've spent with them in Google Research (I am so excited to collaborate with you again very very soon! :-)).

Special shout-out goes to Alex—for being so supportive, for helping me so much with my job applications, and for his overall great attitude. And to Ed—who believes in me, and who seamlessly opens doors for me. Also, a thank you to a great co-author and collaborator: Jaya Kawale.

I would not be in the place of writing this thesis, let alone be in the US, or do a PhD, had it not been for my sister Evangelia. All the great things that have happened to me in the last five years are a direct result of her existence alone. Evangelia, thank you for being the most amazing sister and friend a person could ask for (B.F.A.S.F.E !!!); thank you for our great years at our minneapolis home; for constantly inspiring me; for giving me the best ever reviewer's comments on my works; for all the great things we've done together (trips, going out, coffee shop studing, cooking classes, our dinners with Netflix, ...); and overall, for your tremendous love and support. You've made my whole transition to the US and to "adult life" so much easier and more enjoyable. I am so happy that we've spent these years together and I am looking forward to where our next adventures might take us (company! collaborations!). I am incredibly lucky I've got you; I hope you know how much I love you!!

A major thank you goes to my parents, my mum Eleni and my dad Andreas; to whom I obviously owe everything, including being here writing this thesis. Mama, thank you for your unbelievable love and support throughout all my life, and for believing in me—I love you so much. Thank you for being my best and first machine learning student and for having read and given me feedback on everything I've ever written! For being apart from an amazing mum, a really great friend who is the best listener; for our travel adventures throughout the world. I know that whatever happens, wherever I will be, you'll be there for me—and this gives me great strength. Baba, thank you for making these five years so much easier for me, just with your positive attitude and smile, for always showing me how much you love me and you care for me, for your hug that is always waiting open for me—I love you very much. Also, a thank you goes to my Nounos Giorgos, for always thinking of me and caring for me.

I want to make sure I thank all my friends, who have made my (PhD) life so much more enjoyable. Agi, Ioanna, Niko, Andrea—you've been so amazing to my sister and me, you've been so nice and sweet and caring that I am very lucky I have you in my life! I will miss you very much, but I am sure we will meet again further down the line. Thank you to the "greek gang" (Maria, Vassilis, Dimitris, Nikos, Panos, Charilaos), the "honorary greeks" (Fateme, Yanning, Donghoon), the "gym girls" (Ancy, Maria, Agi), the “starbucks crew” (Panos, Dimitris, Vasilis, Giannis)—I am very lucky I’ve met you, and we’ve spent time together; I will miss you all! Also, I want to thank my best friends in Greece: Maria, Foteini, and Alexandra—I’ve missed you throughout these years and I am so grateful I know that whenever I come back, our conversations are as if no day has passed. Also, thanks to all my friends during the internships, who have made my summers that much happier, and to all the friends I’ve made during my travels to conferences (e.g. Eleni, Giorgos, Diana, Adonia, etc.). I am sure I’ve missed many people I should have thanked (thank you, and forgive me!).

Also, special thanks to my undergraduate advisor, Prof. George Moustakides, for having great belief in me, and for shelflessly supporting me; giving me life and career advice even during the PhD. To my mentor in my first job ever, Matthieu Mayran, for giving me my first opportunity to work in Google, and opening a door of possibilities for me. Also, a big thanks to all the great people I’ve met during my interviews and who have supported me offering me great opportunities.

Finally, a thank you to my lab mates throughout these years: Sheng, Huahua, Igor, Sijie, Yingxue, Xinyan, Puja, Farideh, Nick, Karthik, Robert, Vidyashankar, Soumyadeep, Qilong, Andre, Chintan, Hardik, Xiaoli; you’ve all been so nice and supporting, thank you!!

Overall, I want to thank everyone who has played a big or smaller role in my life during doing the work on this thesis; I am grateful for all your help and support, one way or another. And I am looking forward to the next chapter of my life (hopefully, full, happy, and exciting)!

Dedication

To Evangelia, Mum, and Dad.

Evangelia: for being the best sister and friend ever, and the most inspiring person

Mama: for your unbelievable love and belief in me

Baba: for always making me feel like your little girl

—I love you very much! And I am so lucky to have you!!

Abstract

Recommendation systems have become an integral part of our everyday lives. Although there have been many works focusing on recommendation quality, many real-world aspects of the recommendation process are typically overlooked: How can we ensure that the very top recommendations users see are engaging? How to recommend venues matching user interests, while preventing many users from being directed to the same venue? Can we design recommenders which first converse with users and then give a recommendation? What is the best way to model recommendation systems as interactive systems, while learning on-the-fly the user-item structure? To what extent can a malicious party perform machine learned adversarial attacks against a recommender?

The goal of this thesis is to pave the way towards the next generation of recommendation systems tackling such real-world challenges to improve the user experience, while giving good recommendations. This thesis, bridging techniques from machine learning, optimization, and real-world insights, introduces novel tools to address the above questions. We focus on three directions: (1) encoding real-world constraints into the objective functions, (2) learning to interact with users, and (3) modeling machine learned fake users with malicious goals.

For the first direction, by adjusting the optimization objective to capture real-world constraints—(1a) the screen space is small, creating the need for the top recommendations to be relevant, (1b) the item capacities are limited—we suitably guide the learning of model parameters. For the second direction, to balance the need to explore users' preferences with the desire to exploit what has been learned, at a large user and item scale, we combine interactive learning techniques with the principle that similar users tend to behave similarly. This combination results in novel recommendation systems that learn to (2a) converse with new users, and (2b) collaboratively interact with users. For the third direction, taking the perspective of an adversary of the recommender, we use machine learning to learn fake user profiles, which are indistinguishable from real ones, while having a malicious goal. Illustrating the vulnerability of modern recommenders to machine learned attacks will arguably create new directions for designing robust recommendation systems against such attacks.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The focus of this dissertation is on how to *create better personalized user experiences using machine learning*. Our goal is to expose the reader to novel views of the recommendation problem capturing real-world challenges which have been largely overlooked by the recommendation literature, and to introduce machine learning tools to approach each view. The work presented in this dissertation can be used as a stepping stone to transition to the new generation of real-world recommendation systems.

Why Care About Recommendation?

Everyday life is filled with choices—which movie to watch? what articles to read? which pair of shoes to buy? which restaurant to try out?; the list goes on and on. The role of recommendation systems is to assist the user with making these choices, by curating for the user a list of things they would be interested in. This role has become increasingly important, due to the plethora of choices available and the creation of new content every hour of the day.

It is not surprising then, that entire industries (Netflix, Pandora, Amazon, to name a few) have been built harnessing algorithms with the goal of providing good recommendations. With all the industrial and research interest in the recommendation problem [7, 57, 94], one could think that this is a solved problem. This is far from the truth. There are many facets of the recommendation process which have received little to no attention. Thus, there is large untapped potential for improved user experience, that this thesis attempts to shed light on.

When doing recommendations, as expected, the things that one thinks are great, can be very different from what another person likes. As a result, the experience one user has with the system could be potentially very different from the experience of another user. This makes *personalization* important. To achieve good personalization at scale, machine learning algorithms are fundamental.

Why Machine Learning?

Imagine a computer program that would hand-code, say: if Bob likes french documentaries and "Die Hard"-type movies; has poorly rated romantic comedy movies; and is between 25 and 35 years old, then, a good movie to suggest for his next Friday movie night is a "James Bond" one that he has not watched before.

It is easy to see that such an approach would not scale, as enumerating all such possibilities for hundreds of thousands of users and movies is infeasible.

This is where machine learning comes in.

The field of machine learning harnesses the power of data, which is fortunately available thanks to the advent of the Web. Using the data, machine learning algorithms attempt to understand the underlying data patterns, and *learn* the program, i.e., the mapping of some input to some output. In the above example, the input would be Bob's data along with all other system users' data, and the output is recommending the James Bond movie to Bob. To accomplish this task, machine learning algorithms rely on three things: (1) data, (2) model, and (3) loss functions to optimize.

Data.

We start with the data. When trying to classify an image as a cat or a dog, the data is tuples of (input: pixels of the image, output: dog/ cat). In recommendation systems, a very important piece of data is how different users have rated or interacted with different items, or else (user, item, rating) tuples. *Items* is an umbrella term covering movies to watch, news articles to read, products to purchase, and so on, depending on the application domain that the system is tailored for.

Of course, other data could be available beyond ratings, such as proprietary information about the user (e.g. age, gender, location, social network among users), and about the item (e.g. genre, date of release, price, thematic content). For simplicity, throughout this thesis, we assume the presence of only (user, item, rating) tuples, succinctly represented as a sparse matrix with the users as rows and the items as columns, and the rating/ interaction values as matrix entries. The matrix is sparse, as only a few items have been *explicitly* rated (e.g. in a 1-5 rating scale) or *implicitly* rated (e.g.

clicked/ viewed) by each user.

Model.

The model encodes the beliefs about the data patterns. A prominent model we will use throughout this dissertation is the one of *collaborative filtering* [94, 150]. This term denotes the assumption that similar users tend to like or rate things in a similar way. One can try to realize this purely based on observed correlations. For the above example, the model would need to learn underlying facets of the movies (capturing e.g. obscure documentaries, French movies, action movies, movies liked by around 30 year-olds), and of the users (capturing age, liking french movies, and so on) (Chapter 2).

Loss Function—*or encoding real world constraints.*

The loss function captures how good of a fit the algorithm's predictions are (what does the recommendation algorithm predict or suggest to each user?) compared to the ground truth (what does each user actually like?). In particular, the machine learning algorithm attempts to learn the model parameters, so that the loss function measuring the goodness of fit of the model to the ground truth, both in the data used for the algorithm's training, and in some held-out set of unseen rating tuples, is minimized. This ability to *generalize* well in previously unseen data is the core of machine learning; typically, this is realized by the model chosen, the prior beliefs, the given representation of the data, and the controlling of the model's complexity [70].

What the loss function does, is that it guides how the learning of the model parameters proceeds. In 2006, when the Netflix prize happened [23], (a competition by Netflix largely responsible for boosting the interest in recommendation, which released data inviting teams to develop algorithms surpassing the recommendation performance of their baseline), the recommendation problem was posed as a *matrix completion* problem. That is, the task was to predict with how many stars a user will rate a movie, penalizing mistakes using the square loss.

However, this view is not well-aligned with how users view and use recommendations. It penalizes, say, 5-star movies predicted as 3 stars, and 1-star movies predicted as 3 stars, equally. In reality, the former is much more important, as a 5-stars movie is one that should appear at the top of the recommendation list. Instead, the problem can be posed as: How can we rank a 5 stars movie higher than a 3 stars movie? Re-iterating, the top of the list is what a user will see; especially as the screen size becomes smaller and smaller (recommendations can now be viewed in a screen as small as the one of a watch or a pair of glasses!) So, an even better formulation is: how can we ensure that *the very top of the ranked list*, instead of the entire list, contains only movies a user will like (say, movies users would have rated with 5/4 stars if shown to them)? We will see in Chapter 3 that this is a better formulation, leading to improved recommendation quality.

The loss function can also encode multiple objectives we care to capture in the user experience [7]. As established before, one important objective is that the items are appropriately ranked for the user, so that they can find something interesting fast. As a result, not surprisingly, most recommendation works have focused on a single objective capturing quality of recommendations [118, 141]. However, other important facets of the recommendation experience exist, such as novelty, or diversity, and have been encoded in the loss functions [115]. In Chapter 4 we bring light to a new objective that we believe is important: that of penalizing the cases when *the expected usage of items exceeds the respective item capacities.* This is important, especially in the case of recommending physical spaces, as recommending a place that ends up being overcrowded, would probably not be a good user experience, if the users have to wait for hours in a queue, or cannot find a ticket. What is more interesting is that we find that such an objective is not necessarily antagonistic to the objective of good recommendation quality.

Part I illustrates how encoding real world constraints into the loss function can indeed lead to the expected changes in the experience users have with the system.

Data Revisited—*or learning to interact.*

In Part I, we assume that the data is magically given to us. But, where does the data actually come from? The short answer is: The recommendation system itself acquires this data, by deciding every time what to suggest to its users. In the extreme, for a system starting to learn to interact with its users, the rating users $\times$ items matrix is empty, and the system decides which ground truth entries to reveal.

To approach the recommendation problem from such a perspective, a radically different view is needed. The recommendation system, as most web systems, *actively interacts* with the user. Every time a user comes to the system, the system *decides* which item, or rather which list of items, to show to the user. Then, the user considers this list and decides, based on some internal model informed by their preferences, and influenced by the list presentation, to say, click on the second item of the list. This now becomes a new data tuple (this user, this item, click), which will be incorporated into the training of the system, and will affect the subsequent system's decisions about this, and about potentially other similar users, too.

One could say that a way of implementing this is by periodically retraining the already learned models, that we will call from now on *offline models*, to incorporate the newly obtained data. This view is not ideal, as apart from the cost of periodic retraining, it misses an important part of the story. That is, every time a user gives feedback, the system knows only what happened with the given system's choice, i.e., whether the user liked something from this particular presented list. It does not know the counterfactual—what would have happened had it shown something else. Such feedback is called limited information or *bandit feedback*, and an entire subfield of machine learning is devoted to it [18, 31]. Put differently, there are two parts to the problem which are coupled: (1) active data collection and (2) learning the model. As a result, recommendation can be naturally viewed as a *learning to interact* problem (else dubbed as *bandit problem*) [104]. Such a problem inherently presents the trade-off: given the limited information, how can we explore the user preference space, while at the same time exploiting what has been acquired about the user needs so far? We will find in both Chapters 5 and 6, that one can use the model's uncertainty about the user preferences as an intuitive way to balance this trade-off.

The bandit learning view can be especially useful when the user is new to the system. Usually, for this scenario, recommendation systems use external features representing the user so to better capture the profile, and relate this profile with that of existing users. However, what if such feature information is not available? Conceptually, if in your everyday interactions with people, a person you don't know asks you for a recommendation, typically, you would first ask a couple of questions to better understand what this person likes or not, and then you would give a recommendation. Motivated by this, Chapter 5 presents a shift towards more conversational recommendation systems—asking the user a few questions and then giving a recommendation. With the recent surge of interest in personal assistants [15], viewing recommendation as a conversation is particularly interesting, and models such as the ones developed in Chapter 5, as well as in our subsequent work in [49], could prove useful for better eliciting user preferences, and increasing the user's engagement with the system/ assistant.

In a nutshell, Part II concerns with which questions to ask the user (Chapter 5), or which items to present to the user (Chapter 6), to deliver a good user experience. Put simply, it focuses on which data to ask from a user, while caring for a good personalized user experience.

(Malicious) Data—*or the attack of the fake users.*
A related theme to where the data comes from (Part II) is, whether the ratings used to train the recommendation system are truthful, and reported with good interest. Most recommendation algorithms rely on the assumption that user ratings come from a benign environment. In fact, as previously discussed, they harness the power of collaborative learning across users and items; if Alice and Bob have similar preferences, and Alice likes the movie Pretty Woman, then Pretty Woman would be a good fit for Bob too.

What if though, there is an adversarial party with the purpose to manipulate the recommender's predictions? That is, what if an adversary injects fake user profiles into the recommender's training, so that the underlying similarities learned are disturbed towards some adversarial purpose? In that case, if Bob is similar to some of the fake user profiles, his top-N recommendations could end up being compromised, resulting in a bad user experience.

This is a pressing question which deserves attention, as recommendation algorithms have become such an integral part of social networks, entertainment websites, and information delivery, to name a few. Researchers have been long concerned with this question; however, most robust recommendation systems have been developed having in mind attacks where fake user profiles are hand-engineered [98]. With the recent advances in machine learning, the subfield of adversarial examples has emerged [74]; these works pose finding such examples that are indistinguishable to the human eye from real ones, but nevertheless with high probability to be misclassified, as an optimization problem. Part III seeks out to take a similar optimization-based approach for finding adversarial fake user profiles; although the approach is similar, the setting is considerably more complex than mis-classifying single examples, as the fake users can target entire sub-groups of users or items. The perspective of Part III is from the side of the adversary, not the recommender side; assuming that the recommender does not even know that an adversary is present, i.e., is *oblivious*.

Looking closer, Chapters 4 and 7, although seemingly very different, are motivated by the same question: what happens in the context of a *sharing economy*? Both chapters are a direct derivative of the fact that the recommendations that one person gets might affect the experience of another person, and so on. In the case of Chapter 4, the resources might be scarce because many people would like and are recommended the same item. In the case of Chapter 7, a part of the user population, if carefully crafts an attack using machine learning, can affect other users' experience with the system.

In what follows, we describe in more detail our specific contributions.

1.1 Contributions

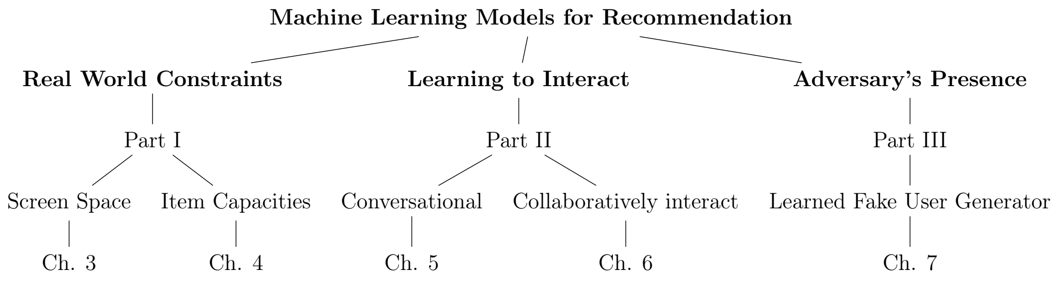


Figure 1.1: Thesis organization: novel views of the recommendation problem.

The thesis addresses a number of important questions regarding how to transition recommendation systems into the wild where real world constraints are present, through the lenses of novel machine learning formulations.

Figure 1.1 gives the overall structure of our research with the mapping to the chapters of this thesis. Each chapter views the recommendation problem from a different angle, motivated by real-world scenarios.

The core contributions of this thesis are five new tools for recommendation systems with real-world constraints. We summarize them at the end of this chapter in Table 1.1. The next subsections provide additional details on the tools and our main findings.

1.1.1 Chapter 3—Recommendation as a Top Ranking Problem

Can we improve the ranking performance at the top of the recommendation lists?

Real-World Scenario: Consider a user interacting with the items suggested by a recommendation system, i.e., which video to watch, or which restaurant review to read. Most users would rarely scroll down to, say, page 9 to see the results. Also, since people

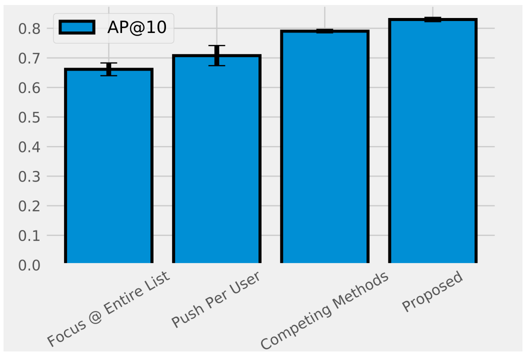


Figure 1.2: New tool for recommendation as a top ranking problem [larger values are better]. Our *proposed collaborative push approaches* outperform in terms of the quality of top-10 recommendations (measured by Average Precision (AP)@10 as defined in chapter 2.4): (i) methods *focusing at the entire list*, (ii) methods pushing down non-relevant items *individually per user*, and (iii) *competing top-ranking* methods optimizing surrogates of ranking metrics.

tend to check their recommendations on their smartphones, only a few recommendations fit in the screen size. Hence, limited space is a common constraint in recommendation, creating the need for recommenders that give good recommendations towards the top of the ranked list.

Classical Tools: The above scenario has led to a conceptual shift from framing recommendation as a rating prediction problem, using square loss to penalize model errors [118], to posing it as a *top-ranking* problem. Most top-ranking approaches have explored the idea based on ranking metrics (2.4) such as NDCG [175] or Reciprocal Rank [159]. However, such objectives are difficult to optimize directly.

New Tool: In Chapter 3, building on advances from the learning to rank literature, we introduce a framework that uses functions paying more emphasis *towards the top of the list*, constructed *collaboratively* across users. Particularly, using low rank representations while working with ranking losses for individual users which focus on accuracy at the top of the list for each user, we offer a flexible new family of approaches for collaborative ranking, dubbed as *collaborative p-norm push*, *infinite push*, and *reverse-height push*. We propose efficient alternating minimization methods for iteratively updating the latent factors, including a new approach for the non-smooth problem based on gradient mapping for minimax problems. Through extensive experiments, we illustrate that: ranking loss indeed performs better than squared loss; collaborative ranking outperforms individual ranking; and the proposed set of algorithms are competitive, usually better than existing baselines (Figure 1.2).

The algorithmic framework and experimental results have appeared as the publication "*Collaborative Ranking with a Push at the Top*" in the World Wide Web Conference (WWW) 2015 [46].

1.1.2 Chapter 4—Recommendation as a Collective Process

How can we give good product recommendations
that do not result in 'out of stock' messages?

Real-World Scenario: Another common constraint in recommendation is that items can be associated with a maximum capacity, such as number of seats in a venue or size of an item's virtual shelf. For example, if the recommendation system suggested most users to go to the same Broadway show or to visit the same theme park, the recommended place would possibly get overcrowded, resulting in long queues.

Classical Tools: Athough other important objectives have been considered, such as novelty [82] or diversity [82], the above scenario has not been addressed in its described form. The closest approaches are those considering constrained resources in different application scenarios, like minimizing volume in email [76], or display advertising with a budget [14]; which however vary significantly from our setting.

New Tool: To close this gap, we develop in Chapter 4 a novel framework for *recommendation with capacity constraints* for item and venue—typically called Point-of-Interest (POI)—recommendation. To the best of our knowledge, we are the first to develop a recommendation algorithm so that the items' expected usage respects the corresponding capacities, while good recommendations are given.

Particularly, we introduce the concept of user propensity to follow the recommendation, and we propose a set of strategies for estimating item capacities and user propensities from data, when they contain no such information. Our experiments in real-world datasets show that adding the *capacity loss* to the recommendation objective, besides improving the extent to which the items' expected usage respect the items' capacities as expected, has the side unexpected benefit of improving the top-N recommendation quality when capacities are proportional to user demand (Figure 1.3).

This chapter is based on the paper "*Recommendation with Capacity Constraints*" which appeared in the proceedings of the Conference on Information and Knowledge Management (CIKM) 2017 [50].

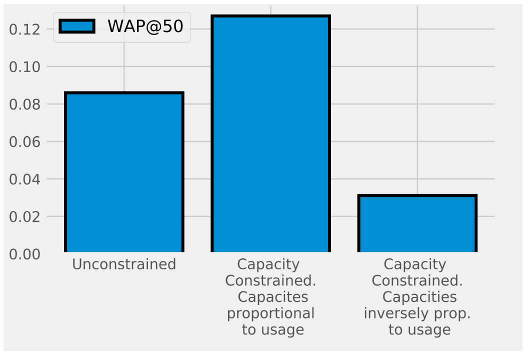


Figure 1.3: New tool for recommendation as a collective process [larger values are better]. Our *capacity-constrained approach* aims at a trade-off among the capacity loss and the recommendation quality. The latter (measured by WAP@50, i.e., a weighted average of the users' AP@50 metrics with weights the tendency of users to follow the recommendations) depends on how item capacities are defined: (i) When item capacities are *proportional to user demand*, recommendations are better compared to *unconstrained methods*; (ii) when capacities are *inversely proportional to usage*, the quality of recommendations reduces.

1.1.3 Chapter 5—Recommendation as a Conversational Procedure

Can we create human-like recommenders
that represent how people give recommendations?

Real-World Scenario: Most recommendation systems behave very differently from a human when asked for a recommendation. In particular, humans can quickly establish preferences when asked to make a recommendation for someone they do not know. For instance, if a conference attendee in your home town, whom you have never met before, asked you for a recommendation on where to eat dinner tonight, most likely you would start with one or two clarifying questions, perhaps whether the person likes seafood, or whether they have a car; and these questions would depend on the context.

Classical Tools: The majority of works on recommendation have focused on the offline problem [94]. Recent works pose the problem in a *contextual bandit* framework [104], but usually rely on the existence of contextual information, which might not always be available. Preference elicitation techniques have been proposed, but typically they either rely on contextual information again, and/ or they are not demonstrated in a fully online setting [40].

New Tool: Motivated by the scenario of recommending to a new user a restaurant to

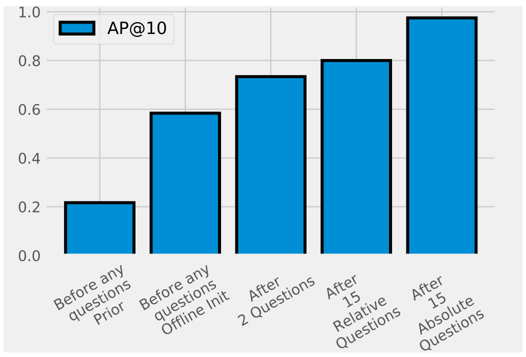


Figure 1.4: New tool for recommendation as a conversational procedure [larger values are better]. Placing the new users as the *average offline user*, performance increases from .217 to .584, even without asking any questions. *After only 2 questions*, performance is improved from .584 to .734 (25% improvement). *Absolute questions* perform better compared to *relative questions*, achieving *after 15 questions* .975 performance.

dine in, in Chapter 5, we present a conversational human-like recommender which asks a few questions, and then gives a recommendation. For this, a multi-faced approach is necessary: studying *large-scale search logs* to understand the space of real user needs and using the insights to collect preference data in a user study; devising elicitation mechanisms and combining them with *active and bandit learning*-inspired question selection strategies. Our contributions are summarized as: (1) We propose a novel view of *human-like recommenders* that converse with new users to learn their preferences. (2) We successfully demonstrate a fully online learning approach for recommendation both using absolute and relative feedback. (3) We propose a systematic approach to incorporating offline data to initialize online learning recommenders, and demonstrate performance improvements, even using weakly labeled offline data. (4) We propose a set of item selection strategies for deciding what question to ask to a cold-start user to quickly infer preferences, and demonstrate benefits of bandit-based strategies. (5) We offer a system that makes effective use of the online user feedback, improving personalized recommendations over a static model by *25%* after asking only two questions. (Figure 1.4)

The work of Chapter 5 has appeared as the paper "*Towards Conversational Recommender Systems*" in the proceedings of the Knowledge Discovery and Data Mining Conference (KDD) 2016 [51].

1.1.4 Chapter 6—Recommendation as Learning to Interact with Users

Which interactive recommendation systems are the most effective?

Real-World Scenario: Most recommendation systems build an offline model based on past user-item observations that is periodically retrained to incorporate new observations. Besides the computational overhead of retraining, this approach is limiting as it has been shown there can be a big gap between offline and online metrics measuring user engagement [7]. To capture online user behavior and optimize for long-term user engagement, a promising direction is to design interactive recommenders that continuously learn user preferences and balance the trade-off of exploiting what the model has learned about the user so far, versus giving a chance to the yet unexplored areas of the preference space.

Classic Tools: While recent literature has explored the idea of combining collaborative filtering approaches with bandit techniques (e.g. [91]), there exist two limitations: (i) they usually consider Gaussian rewards, which are not suitable for implicit feedback data (i.e., clicks, views) powering most recommendation systems, and (ii) they are restricted to the one-item recommendation setting while typically a list of recommendations is given.

New Tool: In Chapter 6 we address both limitations: (i) For one-item recommendation, we devise learning to interact recommendation algorithms for *Bernoulli rewards*, and we provide the first, to the best of our knowledge, overall empirical comparison of the state-of-the-art interactive recommendation algorithms. (ii) For top-N list recommendation following the *cascade* model, we propose the novel framework of *collaborative cascade bandit ranking*, and we show empirically that it outperforms the state-of-the-art in cascade bandit ranking. For both cases, we leverage Thompson Sampling (TS), a technique that naturally balances the explore-exploit trade-off using the model uncertainty. Overall, we find that when the number of users and items is large, propagating the feedback across users and items while collaboratively learning latent features and balancing the explore-exploit trade-off, is the most effective approach for systems to learn to interact with the users (Figure 1.5).

Chapter 6 is published in the proceedings of the SIAM Conference on Data Mining (SDM) 2018 as "*Learning to Interact with Users: A Collaborative-Bandit Approach*" [48].

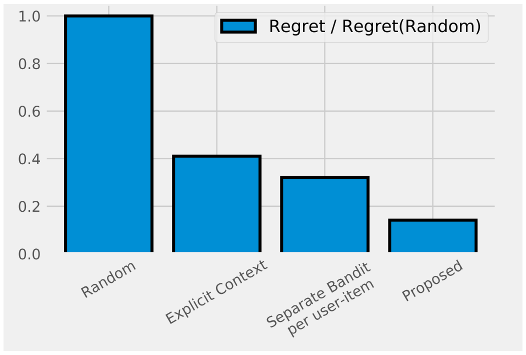


Figure 1.5: New tool for recommendation as learning to interact with users for the cascade list user model [smaller values are better]. Our *proposed approach* of learning on-the-fly the latent user-item structure collaboratively is better than (i) using a *separate bandit per user-item*, (ii) using *explicit context*, and of course the (iii) *random* baseline.

1.1.5 Chapter 7—Recommendation as a Game against Adversaries

How vulnerable are recommendation models to ***machine learned*** *adversarial attacks?*

Real-World Scenario: Inspired by recent developments showing that deep learning models are easy to be fooled [74, 122] in the case of *adversarial examples*, which are found in an optimization setting as adding a minimal perturbation to an existing correctly-classified example, a natural question is: Can we *learn adversarial user profiles* that can deteriorate the recommendation quality of recommendation systems in a principled mathematical framework?

Classical Tools: While researchers have been interested in the direction of adversarial fake user profiles, dubbed as *shilling attacks*, they typically rely on hand-coding such profiles; these are typically profiles that rate with the maximum or minimum score the target item they want to push up or down respectively in users' lists, and rating the rest of the items with scores following the normal or random distribution [98, 127].

New Tool: Instead, in Chapter 7, we explore how to design *machine learned adversarial attacks* and we measure how susceptible latent-factor models are to those. This is an important direction, as finding adversarial users with machine learning is a stepping stone towards evaluating current recommendation algorithms against them, and motivating the need for models robust to adversarial manipulations.

Particularly, we formulate *adversarial recommendation* as a game of an adversary vs. an oblivious recommender; and we solve the problem from the adversary's perspective

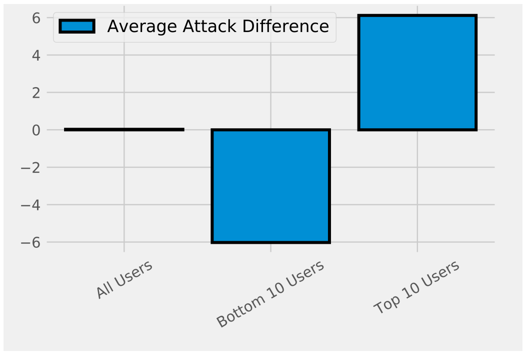


Figure 1.6: New tool for adversarial recommendation. A successful attack for a target item h consists of targeting the user with the higherst predicted score from the recommeder model before the attack. The y-axis shows the average 'attack difference' metric Δ, measuring the magnitude of decrease in the mean predicted score of item h over a set of users. We see that although (i) Δ *over all users* is only .0298; (ii) for the *bottom-10 users*, i.e., the ones predicted before the attack to want h the least, Δ is -6.024—their predicted score increased; and (iii) for the *top 10 users*, i.e., those predicted before the attack to want h the most, Δ is 6.12—their score decreased by $\sim$6.

using: (i) generative adversarial nets [73] to *learn user profiles that mimic the true distribution* and (ii) gradient descent to update them so to *optimize an adversarial goal*. To the best of our knowledge, this is the first time to pose finding fake user profiles in an optimization setting; this allows optimizing various complex objectives in a unified way. We offer a range of experiments evaluating the success of the adversarial fake user generator in a variety of attacks, ranging from targeting the score of a (user, item) entry, removing an item from the top of a user's recommendation list, to targeting the predicted score over a subset of users or items. One main finding of this work is that when the adversary targets the top user of an item, i.e., the user predicted by the recommender model before the attack as the one who wants the target item the most, then the top-K users predicted to want the item, and the top-K items for this top user, are affected as well (Figure 1.6).

This work is in arXiv as "*Adversarial Recommendation: The Attack of the Learned Fake Users*" [47] and is currently under review.

1.2 Broader Impact

Follow-up works have extended our reverse height collaborative push construction (Chapter 3, [46]) for various useful recommendation scenarios, such as for cross-domain recommendation [137], for generating trust-based recommendations [138], for incorporating social relationships [136], for semi-supervised settings [22]. Also, application domains such as venue suggestion [13], or fine-grained tweet geolocation [41], seem to benefit as well from the collaborative height formulation.

Our conversational recommendation work (Chapter 5, [51]) has been considered as a starting point for transitioning towards more conversational systems [155, 135]. It has been included as a *suggested reading* in graduate classes (Illinois CS 591txt Fall 2016 Text Mining seminar, BRAIN York Data Mining Seminar 2016). Also, it has been featured as a contributed talk in the Women in Machine Learning workshop 2016, co-located with NIPS.

The gist of most works presented in this dissertation has been included in two invited presentations in the AI With The Best conference [44, 45].

1.3 Bibliographical Notes

For the theme of ranking with a focus at the top, motivated by the need for scalability of our top-ranking methods, in a subsequent work in collaboration with Mojtaba Kadkhodaie, Maziar Sanjabi, and my advisor Arindam Banerjee, we developed a general, provably and empirically fast optimization method which accelerates the large-scale optimization method of Alternating Direction Method of Multipliers [28], and we showed that the performance of our method on ranking problems for web search can surpass specialized methods.

Regarding the theme of conversational recommendation, in a subsequent joint work with Alex Beutel, Rui Li, Sagar Jain and Ed Chi, titled "*Q&R: A Two-Stage Approach toward Interactive Recommendation*", we developed a two-stage approach in order to further engage users in YouTube. We observed that, factoring recommendation into the two stages: (1) "what to ask?", and (2) given the user's reply, "how to adapt the recommendations?", can lead to a better understanding of user preferences, and as a result, better user experiences in multiple applications.

These two works have not been included in this thesis, but can be found in the proceedings of the Knowledge Discovery and Data Mining (KDD) conference 2015 [88],

Concept	Classical tools	New tool
Focus at the top of the recommendation list.	Algorithms optimizing square loss [156], or surrogates of non-convex ranking metrics [159, 175].	Family of algorithms for more emphasis on the top of the list, using **push** at the **collaborative heights**. Ch. 3; [46]
Respect the limited capacities of the items, while giving good recommendations.	No prior tool for the exact scenario. Other objectives (e.g. serendipity, novelty, diversity) have been considered [115]. Other resource-constrained application domains are related [80].	**Capacity-Constrained** recommendation, based on capacities and users' propensities to listen. Ch. 4; [50]
Converse with users to understand their preferences before giving recommendations.	Offline, non-interactive, models [119]. Interactive models relying on contextual features, which might not be available [104]. Preference elicitation techniques, typically using contextual features [40].	Use of **latent factor** recommendation, and **bandit** and **active learning** techniques, in a fully **online learning** setting, asking absolute and relative questions to the users. Ch. 5; [51]
Learning to interact with users.	Focus on Gaussian rewards for the one-item recommendation setting [91], or on using explicit features for the cascade-list recommendation setting [186].	**Collaborative-cascade bandits** for cascade-list setting, and extension of collaborative-bandits for one-item setting for **Bernoulli-type** feedback. Ch. 6; [48]
Find the extent to which machine learned adversarial attacks can affect the recommendation model.	Hand-engineered fake user profiles [98, 127].	Machine learned **adversarial fake user generator**, playing against an oblivious recommender. Ch. 7; [47]

Table 1.1: Tools for novel views of the recommendation process developed in this thesis.

and 2018 [49], respectively.

Next, we present basic background and preliminaries; we then proceed with each of the three main parts of this dissertation: Encoding Real World Constraints, Learning to Interact With Users, and Malicious Data.

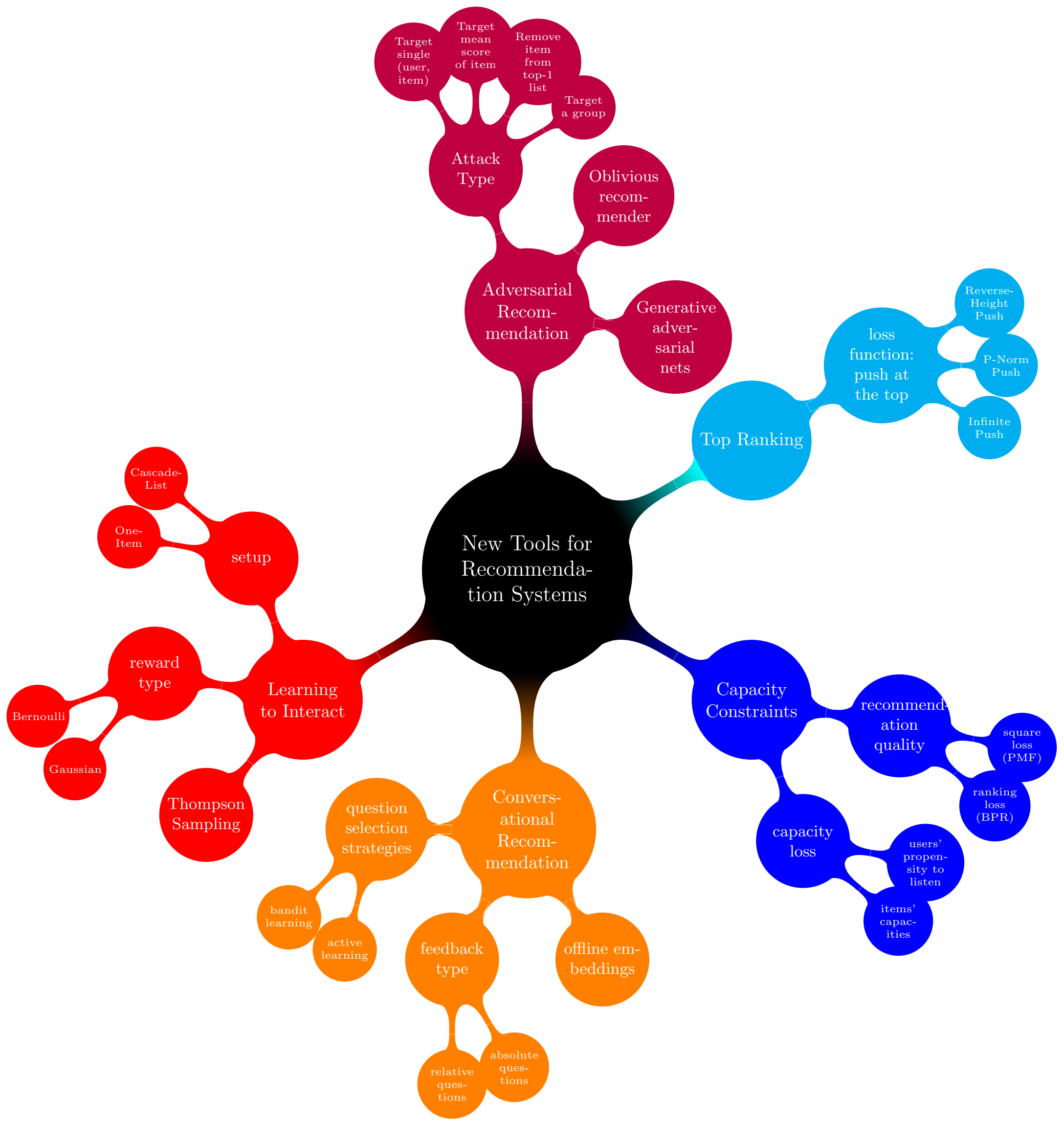


Figure 1.7: Mind map of the key ideas included in this dissertation.

Chapter 2

Preliminaries and Background

We begin with an overview of the background concepts used throughout this dissertation. In discussing such concepts, we cite the general related work, but we will go into more depth in the related chapters of this thesis.

Unless noted otherwise, we will use the following convention. We will use capital calligraphic characters (e.g. $\mathcal{A}$) to denote a set, uppercase characters (e.g., X) to denote a matrix, lowercase bold characters (e.g., $\mathbf{x}$) to denote a vector. All vectors are column vectors, and all row vectors will be represented using the transpose superscipt ^{T} (e.g., $\mathbf{x}^T$). x_{ij} will denote the scalar in the (i,j) position of matrix X. We will use $\mathbf{x}_i^T$ to denote the i-th row of matrix X, and $\mathbf{x}_j$ for the j-th column of matrix X. We will use $\hat{\cdot}$ to denote predicted values, e.g., $\hat{x}_{ij}$ is the predicted value for the (i,j) entry of matrix X, and $\tilde{\cdot}$ to denote sampled values. We will use $|\cdot|$ to denote the cardinality of a set.

2.1 Basic concepts

A recommendation system is broadly defined as a system assisting the user by tailoring for them suggested lists of, say, products they might want to purchase, songs they might want to listen to, news articles they might like to read, and so on.

In a recommendation system there is a set of *users* $\mathcal{U}$, a set of *items* $\mathcal{I}$ (products/ news articles/ songs/ venues, etc.), and *feedback/ interaction data.* The data represents the *feedback* users from $\mathcal{U}$ have given on items from $\mathcal{I}$, or more generally *user-item interactions.* As discussed in the introductory chapter, additional data might be available, such as contextual feature data, describing the users and/ or items; but, this thesis will rely only on feedback data. The only exception will be chapter 4, in which we will use the additional information of geographical location about the items.

Types of feedback. The feedback can be *explicit*, i.e., the user explicitly says that they like an item, or dislike an item. For example, the user clicks thumbs up or thumbs down in a song, or rates a movie in a scale of say, one to five stars, with more stars denoting higher liking. Usually, explicit feedback data, are either binary, represented as +1, and -1 (or 0), for positive and negative feedback respectively, or they are in a multi-level relevance scale, e.g. the 1, 2, 3, 4, 5 rating scale.

Alternatively, the feedback can be *implicit*, i.e., derived from the browsing behavior of the user—e.g. the user clicked on a product, checked in a venue, or viewed an article. Implicit feedback data are represented as positive values, usually either 1s, representing that an interaction has happened, or positive integers denoting the number of interactions (e.g. the number of times the user has checked in the venue). Importantly, implicit feedback data do not contain negative observations; so, several strategies sampling negatives from the unobserved user-item interactions have been proposed [129].

A further distinction of feedback data is: the feedback can be *absolute*, e.g., something is liked or disliked by the user, or *relative*, e.g., user i prefers item j over item h, which will be represented as $j >_i h$.

In chapter 3 we use binary explicit feedback data, denoted as $+1, -1$ for relevant and non-relevant items, with a relative view-point—relevant items should be preferred over non-relevant items. In chapter 5 we use explicit absolute binary feedback, i.e., $+1, 0$ representing like, dislike, and explicit relative feedbcak, represented as tuples (i, j, h) noting that $j >_i h$. In chapters 4 and 6 we experiment with both implicit and explicit feedback. Last, in chapter 7 we train the recommender under attack with explicit feedback in a rating scale of 1 to 5. For ease of presentation, in what follows, we will assume absolute feedback data, unless otherwise noted.

Scoring matrix (alternatively called *rating/ interaction matrix.*) The data, typically given as tuples of (user i, item j, rating), can be succintly represented with the mathematical construction of a matrix R, where users are the rows, and items are the columns (Figure 2.1). Let $M = |\mathcal{U}|$ be the number of users and $N = |\mathcal{I}|$ be the number of items. Then, the matrix $R \in \mathbb{R}^{M \times N}$ is sparse, as every user i has rated only a few items $\mathcal{L}_i \subseteq \mathcal{I}$, and every item j has been rated by a subset of all users $\mathrm{Ra}(j) \subseteq \mathcal{U}$. Formally, a row $\mathbf{r}_i^T$ represents user i's ratings with only $|\mathcal{L}_i|$ non-zero entries, and a column $\mathbf{r}_j$ represents

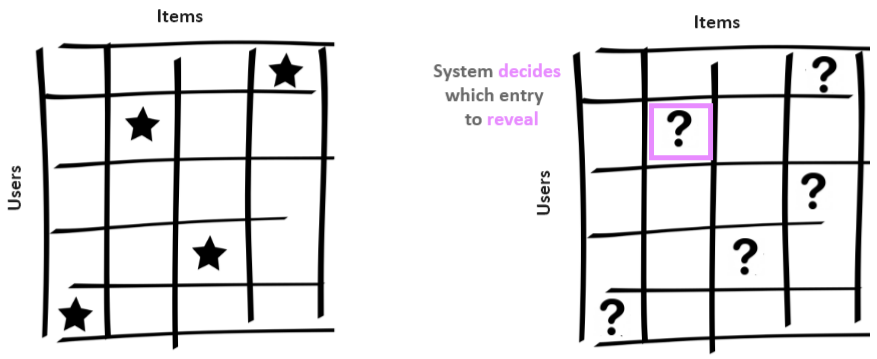


Figure 2.1: Sketch of a users by items scoring matrix. The *left panel* represents the offline model view, while the *right panel* illustrates the interactive learning view.

the ratings on item j with only $\mathrm{Ra}(j)$ non-zero entries. The zeros represent missing data.

Recommender strategies. The wide interest in personalized recommendations has sparked substantial research in this area [94]. The most common approaches are *content-based* approaches [133] (which rely on the availability of external feature information to build user and item profiles) and *collaborative filtering (CF)* [57, 118]. In this thesis, we build on the collaborative filtering principle, which powers most modern recommenders.

Collaborative Filtering (CF) relies on the intuitive idea that similar users tend to like different items similarly. Collaborative filtering methods use an a-priori available set of user-item ratings to learn the interdependencies among users and items. They predict a user's rating on an item either via the neighboring items' ratings (*neighbor-based* [57, 150, 42]) or by inferring latent factors which find similar users and items in a low dimensional embedding (*latent factor-based* [118, 160, 43]). In this dissertation we build on top of the latent-factor based approach [94] (more details in Section 2.2).

Offline models vs. Learning to interact. Typically, recommendation algorithms are built with the view of fitting their model, say a collaborative filtering one, on given feedback data (Figure 2.1, left panel). However, this view can only allow to learn associational relationships among the data—for example it can learn by observing the past data that when user i likes item j, and user i' who is similar to i likes items j, k, then user i will probably like item k, too. However, such a view cannot answer questions of the form *what would happen if we show item j' on user i*? To answer such questions, we cannot have a static (offline) perspective of the data anymore.

Instead, we need *online* type of learning, and we need the ability to make an *intervention*, aka. intervene and show to user i item j'; get i's feedback on item j' (in terms of clicks, time spent, or other signals); and update the model with the newly acquired data. Recently, there has been a line of work on recommendation systems modeling them as interactive systems instead of fitting offline models [104, 85]. While chapters 3 and 4 perform offline model fitting, chapters 5 and 6 represent examples of such a shift toward modeling recommendation as an interactive process.

Bandits. Importantly, when you are doing an intervention, showing item $j^{'}$ to user i, you only get to see feedback for this i-$j^{'}$ interaction; namely, you do not get access to the counter-factual: what would have happened had you shown something else, e.g. an item j''? Thus, this is a setting where only *limited information*, or else *bandit feedback*, is available (in contrast to the classic *online learning* scenario occuring e.g. in weather forecasting, where full information is observed after the prediction [31]). This makes the construction of a *Multi-Armed Bandit*, i.e., a tool for sequential decision making in the presence of an unknown reward distribution whose parameters are to be sequentially learned [152], nicely suited for modeling interventions.

The problem intuitively demonstrates the *explore-exploit trade-off*, i.e., the system has to balance the need to learn new information about the user (*explore*), while focusing on what has already been learned about their preferences (*exploit*). For this, bandit algorithms (e.g. Upper Confidence Bound, ϵ-greedy, Thompson Sampling) have been proposed; a detailed survey can be found in [31]. In chapters 5 and—especially—6, we focus on Thompson Sampling thanks to its good empirical guarantees [38].

Formally, as we will see in more detail in chapter 6, in recommendation, every user can be modeled as a bandit problem, with possible *arms / actions* the items, and with revealed *rewards* the clicks / ratings/ other interaction signals. In other words, the interactive recommendation algorithm *actively chooses* which column-entry to reveal for an incoming row-user (Figure 2.1, right panel), while balancing the explore-exploit trade-off, and learning the underlying unknown reward parameters.

2.2 Model: Matrix Factorization

Latent-factor or matrix factorization (MF) based approaches have become very popular in recommendation systems [94] both for implicit [81, 87, 43] and explicit feedback [148]. They predict a user's rating on an item on the basis of low dimensional latent factors, by optimizing an objective function (Section 2.3).

Let $U \in \mathbb{R}^{k \times M}$ be the latent factor matrix corresponding to the users, where the i^{th} column $\mathbf{u}_i \in \mathbb{R}^k$ is the latent factor for user i. Similarly, let $V \in \mathbb{R}^{k \times N}$ be the latent factor for the items, where the j^{th} column $\mathbf{v}_j \in \mathbb{R}^k$ is the latent factor for item j. Simply put, U represents the latent preferences of the users, and V represents the latent attributes of the items. Then, latent factor-based methods model the predicted score of user i on item j as $\hat{r}_{ij} = \mathbf{u}_i^T \mathbf{v}_j$, so that the overall score matrix $R \in \mathbb{R}^{M \times N}$ is of low rank $k \ll M, N$, and can be approximated by $U^T V$. Effectively, this is a *dimensionality reduction* approach, projecting both users and items in the same low dimensional space, where items which are nearby are similar, users which are nearby in the embedding are similar, and items which are near a user are a good fit for this user.

2.3 Learning

Objective functions. In order to learn the model parameters, e.g. the U, V in latent-factor approaches, we need to specify our objective. This objective measures the goodness of fit of the model to the data. Usually the objective is formulated as a *loss function* we want to minimize for the cases when the model's predictions disagree with the data which the model attempts to fit. Our goal is to find the model parameters that minimize the loss function.

One common loss function is the square loss [148, 156], which is used by *Probabilistic Matrix Factorization* `PMF`:

$$\arg\min_{U,V} \sum_{i=1}^{M} \sum_{j \in L_i} (r_{ij} - \mathbf{u}_i^T \mathbf{v}_j)^2. \tag{2.1}$$

As mentioned, the square loss has the drawback of penalizing equally a 3-stars movie predicted as a 1-star, and a 5-stars movies as a 3-stars, which is not what we want. Instead, loss functions which are inspired by viewing the recommendation problem as a *learning-to-rank* problem (i.e., the user is the query, and the items for recommendation play the role of documents to rank pertinent to the query) have been proposed [141, 159]; chapter 3 of this thesis is in fact devoted to merging novel learning-to-rank losses with a latent-factor model. A popular ranking-based loss function is the one of *Bayesian Personalized Ranking* (`BPR`) [141] which focuses on correctly ranking item pairs instead of scoring single items. Let $L_{i,+}, L_{i,-}$ denote the set of positively (+1) and negatively rated (-1) items by user i respectively. Then the objective of `BPR` is:

$$\arg\min_{U,V} \sum_{i=1}^{M} \sum_{k \in L_{i,+}} \sum_{j \in L_{i,-}} \log(1 + \exp(-\mathbf{u}_i^T(\mathbf{v}_k - \mathbf{v}_j))). \tag{2.2}$$

We will use `PMF` and `BPR` as baselines, and will build on them in chapters of this thesis.

Regularization. As our goal is to learn model parameters that do not only fit well the data used for training, but also *generalize* well to new unseen held-out data, we need a way to *prevent over-fitting* of the model parameters to the training data. One prominent way of achieving this is by adding an additional term in the objective function that *penalizes the model complexity.* This is the so-called regularization term. A popular type of regularization is the L_2 regularization, which for latent-factor approaches is:

$$+\lambda\left(\|U\|_F^2 + \|V\|_F^2\right), \tag{2.3}$$

preventing the norm of U, V from becoming too large. The parameter λ controls the extent to which we want to regularize our model, and the symbol $\|\cdot\|_F$ denotes the Frobenius norm of a matrix.

Optimization. To optimize objective functions, a popular approach is to use *stochastic gradient descent (SGD)* [124]. The idea is to modify the parameters by a magnitude proportional to η (i.e., the learning rate) in the opposite direction of the gradient of the objective function. For optimizing the square loss function of `PMF` another popular alternative is to use *alternating least squares*, alternating between fixing U and updating V, for which a closed form update is available, and fixing V to update U with its closed form update [94]. Throughout this thesis we will use SGD updates; we will go more into the specifics in each chapter. For chapter 7 we use alternating least squares to train `PMF`.

Multiple objectives. It has been found that several criteria are important in recommendation systems, beyond just accuracy [97, 143, 1]. So, *Multiple Objective Optimization (MOO)* has been employed to e.g., optimize various ranking metrics [163], diversity [84], novelty [82], time spent along with click-through-rate [9], and more generally to improve user engagement while satisfying a set of desired business objectives [8, 4, 115, 166]. Our work in chapter 4 is an example where multiple objectives need to be optimized, given that we want to optimize for expected usage which respects the capacity constraints, while maximizing predictive accuracy.

Bayesian models. An alternative view for learning our model is the Bayesian view. Particularly, we can specify our model as a generative procedure, i.e., a Bayesian model by which we believe our data is generated, and then learn the Bayesian model parameters by maximizing the likelihood of the data being generated [27]. In fact, both `BPR` and `PMF` have a Bayesian interpretation, with the former maximizing the posterior distribution of the probability that a user will prefer the positive items over the negative ones, and the latter maximizing the log-likelihood of the data under the assumption that the ratings r_{ij} are generated by a Gaussian $\mathcal{N}(\mathbf{u}_i^T \mathbf{v}_j, \sigma^2)$, and $\mathbf{u}_i, \mathbf{v}_j$ having spherical Gaussian priors, e.g., $\mathbf{u}_i \sim \mathcal{N}(0, \sigma_u^2 I)$ where $I \in \mathbb{R}^{k \times k}$ is the identity matrix.

We will adopt the Bayesian view in chapters 5 and 6, as this will allow us to take advantage of the model uncertainty, as captured by the variance in the parameter distributions.

2.4 Evaluating Recommendation Performance

For user i, we obtain the user's predicted recommendation list by sorting all items by decreasing values of predicted ratings $\hat{\mathbf{r}}_i$. We evaluate this list by looking at the ground truth $\mathbf{r}_i$, i.e., capturing whether the user liked/disliked each item, or with how many stars did the user rate each movie. We will denote with *top* the truncation threshold, reflecting the number of recommendations we want to consider. Let ℓ be the index ranging over positions in the ranked list ($\ell = \{0, \ldots, top - 1\}$), and let $[\ell]$ represent the index of the item present in rank ℓ, with $[\ell] = 0$ corresponding to the index of the top recommended item. We will denote with $r_{i[\ell]}$ the ground truth relevance of the item in the ℓ-th position, for the user i.

In what follows, we describe evaluation metrics capturing the performance of a recommender system that we will use in chapters of this dissertation.

We start with two metrics widely used in learning-to-rank [29, 37, 99] which are used to capture the quality of recommendations at the top of the list, i.e., Average Precision @*top* (*AP*@*top*), and Normalized Discounted Cumulative Gain@*top* (*NDCG*@*top*).

Average Precision (AP)@*top*. *AP@top* is a widely used, precision-oriented metric [77] for capturing accuracy in the top. *AP@top* is defined as the average of precisions computed at each liked position in the *top* items of the user's ranked list. *P*@ℓ (*Precision*@ℓ) is the fraction of liked items out of the top $\ell + 1$ ranked items. Thus,

$$AP@top = \sum_{\ell=0}^{top-1} \frac{P@\ell \cdot r_{i[\ell]}}{\min(top, \#\text{ of liked items})} \tag{2.4}$$

where $r_{i[\ell]}$ can be either 1 (relative) or 0 (non-relative) to the user.

Normalized Discounted Cumulative Gain@*top* (DCG@*top*). Discounted Cumulative Gain@*top* (*DCG*@*top*) captures the importance of finding the correct ordering among higher ranked items compared to that among lower ranked items, by discounting the importance weight with the item's position in the ranked list. *DCG*@*top* is formally given by

$$DCG@top = \sum_{\ell=0}^{top-1} \frac{2^{r_{i[\ell]}} - 1}{\log_2(\ell + 2)} \tag{2.5}$$

NDCG@*top* is the ratio of *DCG*@*top* to the Ideal *DCG*@*top* for that user, i.e the *DCG*@*top* obtained had the user's recommendation list been sorted according to the ground truth ratings of the user.

After computing the metrics per user, we average the results over all users, and report the results for *Mean Average Precision*@*top* (MAP@*top*) and *Mean NDCG*@*top*. Higher values, closer to 1, are better.

Next, we discuss two metrics capturing quality of predictions in the entire list, namely Root Mean Square Error (RMSE), which was popularized by the Netflix challenge but it is accompanied by the problems of approaching recommendation as a matrix completion problem mentioned in the introduction; and 0/1 Pairwise loss which captures whether relevants items are ranked over non-relevant ones throughout the whole list.

Root Mean Square Error (RMSE). RMSE measures test set rating prediction accuracy in a set of held-out user ratings, for each user i, $L_{i,\text{test}}$:

$$\text{RMSE} = \sqrt{\frac{1}{M}\sum_{i=1}^{M}\frac{1}{|L_{i,\text{test}}|}\sum_{j\in L_{i,\text{test}}}(\hat{r}_{ij}-r_{ij})^2}. \tag{2.6}$$

0/1 Pairwise Loss measures the average number of incorrectly ordered pairs (-1 ranked above +1):

$$\text{0/1 Pair. Loss} = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{|L^-_{i,\text{test}}|\cdot|L^+_{i,\text{test}}|}\sum_{j\in L^-_{i,\text{test}}}\sum_{k\in L^+_{i,\text{test}}}\mathbb{1}[\hat{r}_{ij}\geq \hat{r}_{ik}], \tag{2.7}$$

where $L^+_{i,\text{test}}$ and $L^-_{i,\text{test}}$ denote the set of positively and negatively rated items by user i in the test set respectively, and $\mathbb{1}[\cdot]$ denotes the indicator function, or else the 0-1 loss.

For RMSE, and 0/1 Pairwise Loss, smaller values, closer to 0 are better.

Cumulative regret. When viewing recommendation as a learning to interact problem, we need a metric measuring the online algorithm performance in a fixed time horizon T [31]. This metric measures the gap between the best the algorithm could have done in hindsight, and the actual model performance—smaller values closer to 0 are better; a formal definition of regret will be given in chapter 6.

2.5 Table of symbols

In Table 2.1 we list common symbols used in the thesis. Each chapter then also introduces chapter-specific concepts and symbols.

2.6 Table of datasets

Each chapter of this thesis uses a subset of the following described publicly available real-world recommendation datasets: four movie datasets (MovieLens 100K, MovieLens

Symbol	Description
M (or m)	Number of users in the recommendation system
N (or n)	Number of items in the recommendation system
R	Scoring/ Rating/ Interaction matrix, $R \in \mathbb{R}^{M \times N}$
i	User in the recommendation system, i.e., $i = 1, \dots, M$
j	Item in the recommendation system, i.e., $j = 1, \dots, N$
$\mathcal{U}$	Set of users, $\mathcal{U} = \{u_1, \dots, u_M\}$
$\mathcal{I}$	Set of items, $\mathcal{I} = \{i_1, \dots, i_N\}$
$\lvert \cdot \rvert$	Cardinality of a set
$\mathcal{L}_i$	Subset of items rated by user i, $\mathcal{L}_i \subseteq \mathcal{I}$
$\mathrm{Ra}(j)$	Subset of users who have rated item j, $\mathrm{Ra}(j) \subseteq \mathcal{I}$
k (or d)	Rank of the latent factors
U	Latent factor of users, $\mathbb{R}^{k \times M}$
V	Latent factor of items, $\mathbb{R}^{k \times N}$
$\hat{R}$	Model predicted score matrix, e.g. for latent-factor, $\hat{R} = U^T V$
$\mathbf{u}_i$	Latent factor for user i, i-th column of U, $\mathbf{u}_i \in \mathbb{R}^k$
$\mathbf{v}_j$	Latent factor for item j, j-th column of V, $\mathbf{v}_j \in \mathbb{R}^k$

Table 2.1: Table of common symbols used in this thesis.

1M, Netflix, EachMovie), a musical artist dataset from Yahoo! containing user ratings of music artists [174], and two datasets containing user check-ins[1] in Points-of-Interest (POIs). Each chapter will describe the specifics on the evaluation setup used, and how the data are transformed to best fit the application scenario. Below a general pre-processing of the datasets used in all chapters is described.

The two MovieLens datasets were kept in their original form. The EachMovie dataset used was a subset of the original EachMovie including the ratings of 30,000 randomly selected users with 20 or more ratings[2]. The Yahoo! dataset was subsampled to speed up the experiments, since the original dataset had close to 2 million users and 100,000 items. For the Yahoo! dataset, we first selected the 1,000 most popular items and then selected the 6,000 most heavy users (with the most ratings). In addition to the subsampling we removed the user ratings of the special value 255 (‘never play again’) and rescaled user ratings from 0-100 to the 1-5 interval. The rescaling was done by mapping 0-19 to 1, 20-39 to 2, etc. The procedure followed for the Yahoo! dataset is close to the one described in [170]. Similarly, for the Netflix dataset, we considered a dense sub-matrix containing the 500 most popular items and the 5,000 most heavy

[1] http://www.ntu.edu.sg/home/gaocong/data/poidata.zip [2] http://mlcomp.org/datasets/350

Dataset	Number of Users	Number of Items	Number of Ratings
MovieLens 100K	943	1,682	100,000
MovieLens 1M	6,040	3,706	1,000,209
Netflix	5,000	500	1,922,815
EachMovie	30,000	1,623	2,109,780
Yahoo! R1	6,000	1,000	3,522,232
Foursquare	2,025	2,759	85,988
Gowalla	7,104	8,707	195,722

Table 2.2: Table of common datasets used in this thesis.

users. For the Gowalla and Foursquare datasets, we removed the users and POIs with less than or equal to 10 ratings.

The statistics of the datasets (total number of users, items, and ratings) are shown in Table 2.2, and for ease of reference, each chapter will report the statistics of the respective subset of datasets it uses.

This concludes the presentation of the basic concepts needed to follow the contributions presented in the main body of this dissertation. Every chapter will also contain its own review of chapter-specific related works as needed.

Part I

Loss Functions: Encoding Real World Constraints

Based on work previously published in
WWW 2015 [46] and CIKM 2017 [50].

Part I: Overview

To transition recommenders to the real world, we need to design loss functions that encode the real-world constraints imposed by the application scenario. Here, we examine two such constraints: (1) The constraint of screen space in which the recommendations need to fit—making it necessary for the absolute top of the recommendation list to be relevant for the user (as this is what they will get to see). (2) The constraint of every item candidate for recommendation having a certain limited capacity (i.e., number of people who can simultaneoulsy use the same product or visit the same place)—making it as a result important to not recommend to everyone the same product, venue, etc.

In pursuit of the goal of encoding real-world constraints in the recommendation algorithm's objective function, we offer the following contributions:

- **Collaboratively push relevant items to the top of the list.** Or push down non-relevant items from the top of the list; or push down the highest ranked non-relevant item from the top of a user's list, *collaboratively* across users and items.
- **Penalize expected usage of items exceeding respective capacities.** Introducing the notion of a user's propensity to listen—or more simply, user's tendency to follow a system's recommendations, and using items' capacities as external information—, we formalize a linear combination of the proposed "capacity loss" and a classic loss capturing recommendation quality.

In both cases, we find that a suitable loss function can successfully encode the respective real-world constraint, i.e., it can lead to (1) a better quality of recommendations at the top of the list, and (2) a minimized capacity loss, while achieving improved recommendations when items' capacities are proportional to demand.

Chapter 3

Recommendation as a Top Ranking Problem

Can we improve the ranking performance at the top of the recommendation lists?

The goal of collaborative filtering is to get accurate recommendations at the top of the list for a set of users. From such a perspective, collaborative ranking based formulations with suitable ranking loss functions are natural. While recent literature has explored the idea based on objective functions such as NDCG or Average Precision, such objectives are difficult to optimize directly. In this chapter, building on recent advances from the learning to rank literature, we introduce a novel family of collaborative ranking algorithms which focus on accuracy at the top of the list for each user while learning the ranking functions collaboratively. We consider three specific formulations, based on collaborative p-norm push, infinite push, and reverse-height push, and propose efficient optimization methods for learning these models. Experimental results illustrate the value of collaborative ranking, and show that the proposed methods are competitive, usually better than existing popular approaches to personalized recommendation.

3.1 Introduction

Probabilistic matrix factorization type of approaches, widespread especially after the Netflix challenge [23], pose the problem of personalized recommendation as a rating prediction or matrix completion problem, trying to predict the ratings of unrated items [5, 148, 156]. Such approaches consider the square loss or variants as a measure of prediction accuracy. In reality, for a 5-point rating scale, predicting a true 5 as a 3 is often more costly than predicting a 3 as a 1, although their square loss is the same. More generally, what matters in a recommendation system is the ranking of items, especially the ranking performance at the top of the list, i.e., the items the user will actually see.

In this chapter, we consider the recommendation problem as a collaborative ranking (CR) problem, such that the ranking loss focuses on accuracy at the top of the list for each user. The consideration is inspired by recent results [20, 170, 175] which support the hypothesis that posing recommendation as a ranking problem, as opposed to rating prediction, leads to better performance.

While existing CR models do optimize certain classical ranking metrics [159, 157, 158, 175], most of these metrics are non convex, so one works with a relaxation of the original problem, e.g., CofiRank [175] optimizes a convex upper bound of a loss based on Normalized Discounted Cumulative Gain (NDCG) [86], CLiMF [159] optimizes a smoothed version of the Reciprocal Rank [19] via a lower bound, etc. Moreover, certain CR models [100] consider the entire list, rather than focusing on accuracy at the top of the list.

We build on two good ideas respectively from the collaborative filtering and the learning to rank literature. First, the low rank representation with latent factors for users and items is indeed a simple and powerful idea. Second, the convex loss functions developed in the recent literature on learning to rank, such as p-norm push, infinite push, and reverse-height push [10, 139, 146], perform well to get accurate rankings at the top of the list. In our work, we use low rank representations for CR while working with ranking losses for individual users which focus on accuracy at the top of the list for each user. The result is a flexible new family of approaches for CR based on collaborative p-norm push, infinite push, and reverse-height push. We propose efficient alternating minimization methods for iteratively updating the latent factors, including a new approach for the non-smooth problem in infinite push based on gradient mapping for minimax problems [124]. Empirical evaluation carefully considers three key questions:

1. Between square loss and ranking loss with focus at the top of the list, which does better with recommendations at the top of the list?

2. Between individual ranking and collaborative ranking, which does better for recommendations across all users?

3. Are the new collaborative ranking methods competitive with related existing

ideas?

Through extensive experiments, we illustrate that ranking loss indeed performs better than squared loss, collaborative ranking outperforms individual ranking, and the proposed set of algorithms are competitive, usually better than existing baselines.

The rest of the chapter is organized as follows. In Section 3.2, we combine the ideas of push and collaborative ranking, and propose three algorithms for collaborative ranking with a push at the top of the list. We empirically evaluate the proposed methods in Section 3.3, review related work in Section 3.4, and give a summary in Section 3.5.

3.2 Collaborative Ranking With a Push at the Top

Consider a collaborative ranking setting with m users $\{\rho_1, \ldots, \rho_m\}$ and n items $\{x_1, \ldots, x_n\}$, where the items can be movies, books, news articles, etc. Each user ρ_i is assumed to have rated a total of $n_i \leq n$ items, of which n_i^+ items are rated as 'relevant' and n_i^- items are rated as 'not relevant.' Let L_i^+ denote the set of relevant items and L_i^- denote the set of non-relevant items for user ρ_i, so that $|L_i^+| = n_i^+$ and $|L_i^-| = n_i^-$. For convenience, for a given user ρ_i, we will denote the relevant items as x_k^+ (not $x_k^{i,+}$ to avoid clutter) and the non-relevant items as x_j^-.

The goal is to construct ranking/scoring functions $f_i(x)$ over the items for each user ρ_i such that relevant items for ρ_i get a higher rank/score. From a ranking perspective, $f_i(x)$ induces a ranking over all items, and the goal is to make sure relevant items are ranked higher than non-relevant items. To accomplish the goal, our algorithms focus on two aspects: the ranking functions $f_i(x)$ for users will be

(i) trained with focus on accuracy at the top of the list for each user, and

(ii) constructed collaboratively across all the users, rather than separately for each user.

The working hypothesis that emphasis on these two aspects for collaborative ranking leads to better performance will be carefully evaluated in Section 3.3, where we consider alternative approaches which give equal importance to the entire list, or train the ranking functions individually on each user. We start with a discussion on how these two aspects are realized in our algorithms.

3.2.1 Push and Collaborative Ranking

Ranking with a Push: The idea of ranking with emphasis on the top of the list has been explored in the learning to rank literature [10, 29, 139, 146]. We build on a specific idea, originally due to [146]. For any user ρ_i, the "height" of a non-relevant item x_j^- is the number of relevant examples x_k^+ ranked below it by the learned ranking function f_i, i.e.,

$$H_i(x_j^-) = \sum_{k \in L_i^+} \mathbb{1}[f_i(x_k^+) \leq f_i(x_j^-)] \ , \tag{3.1}$$

where $\mathbb{1}[\cdot]$ is the indicator function. The vector $H_i \in \mathbb{R}_+^{n_i^-}$ contains the height of all non-relevant items. A high value of $H_i(x_j^-)$ implies that several relevant items are ranked below the non-relevant x_j^- according to $f_i(\cdot)$. The learning of $f_i(\cdot)$ needs to focus on making the heights of non-relevant items small, i.e., *push* the non-relevant items down, with more emphasis on the large entries in H_i, i.e., non-relevant items towards the top of the list. One can consider learning $f_i(\cdot)$ by minimizing the L_p-norm of H_i [146] (p-norm push), or, the L_∞-norm of H_i (infinite push), which simply focuses on the largest element [10]. Other such suitable functions of H_i, which focus on pushing down non-relevant items from the top of the list, can also be considered.

An alternative approach involves pushing the relevant examples up, rather than pushing the non-relevant examples down. For any user ρ_i, the "reverse height" of a relevant item x_k^+ is the number of non-relevant examples x_j^- ranked above it by the learned ranking function f_i, i.e.,

$$R_i(x_k^+) = \sum_{j \in L_i^-} \mathbb{1}[f_i(x_k^+) \leq f_i(x_j^-)] \ . \tag{3.2}$$

The vector $R_i \in \mathbb{R}_+^{n_i^+}$ contains the reverse height of all relevant items. A high value of $R_i(x_k^+)$ implies that several non-relevant items are ranked above the relevant x_k^+ according to $f_i(\cdot)$. The learning of $f_i(\cdot)$ needs to focus on making the reverse heights of relevant items small, i.e., *push* the relevant items up. As before, one can choose objective functions on R_i which put more emphasis on relevant items with high reverse heights, e.g., one can use p-norm or infinite push.

The use of indicator functions $\mathbb{1}[f_i(x_k^+) \leq f_i(x_j^-)]$ to define heights, while conceptually convenient, is not suitable for optimization purposes. As a result, we will use a surrogate for the indicator function. For this work, with $\Delta_i(k, j) = f_i(x_k^+) - f_i(x_j^-)$, we use the logistic loss of the difference as the surrogate:

$$\ell(\Delta_i(k,j)) = \log(1+\exp(-\Delta_i(k,j))) \ , \tag{3.3}$$

noting that it forms a convex upper bound to the indicator function. In what follows, we refer to the height (reverse height) computed using the surrogate, as surrogate height $\tilde{H}_i$ (surrogate reverse height $\tilde{R}_i$).

Collaborative Ranking: Instead of learning the functions $f_i(\cdot)$ separately for each user, we focus on learning them collaboratively. Towards this end, we build on the existing approach of low rank or latent-factor based representation of the scores, widely used in the collaborative filtering literature [118, 175]. Let $U \in \mathbb{R}^{d\times m}$ be the latent factor corresponding to the users, where the i^{th} column $\mathbf{u}_i \in \mathbb{R}^d$ is the latent factor for user ρ_i. Similarly, let $V \in \mathbb{R}^{d\times n}$ be latent factor for the items, where j^{th} column $\mathbf{v}_j \in \mathbb{R}^d$ is the latent factor for item x_j. Then, we assume the scoring function $f_i(x_j) = \mathbf{u}_i^T\mathbf{v}_j$, so that the overall score matrix $F = U^TV \in \mathbb{R}^{m\times n}$ is of rank d, where d is a user specified parameter. Based on such a representation, we have

$$\Delta_i(k,j) = f_i(x_k^+) - f_i(x_j^-) = \mathbf{u}_i^T(\mathbf{v}_k - \mathbf{v}_j) \ . \tag{3.4}$$

Plugging the above form to compute the surrogate height of a non-relevant sample from (3.1) gives

$$\tilde{H}_i(x_j^-) = \sum_{k\in L_i^+} \log\left(1+\exp(-\mathbf{u}_i^T(\mathbf{v}_k - \mathbf{v}_j))\right) \ . \tag{3.5}$$

Similarly, we obtain the form of the surrogate reverse height for a relevant sample from (3.2) as

$$\tilde{R}_i(x_k^+) = \sum_{j\in L_i^-} \log\left(1+\exp(-\mathbf{u}_i^T(\mathbf{v}_k - \mathbf{v}_j))\right) \ . \tag{3.6}$$

In the rest of the section, we discuss three different loss functions which work with such surrogate heights and put emphasis on accuracy at the top of the list. We outline algorithms for minimizing such loss functions over the latent factors (U, V).

3.2.2 P-Norm Push Collaborative Ranking

The main idea in p-norm push collaborative ranking (`P-Push CR`), building on [146], is to consider the L_p norm of the surrogate height vector $\tilde{H}_i$ with elements as in (3.5), and construct an objective function as a suitable scaled sum of such norms over all users. In particular, we consider:

$$\begin{aligned}\mathcal{E}_{\text{p-push}}(U,V) &= \sum_{i=1}^{m} \frac{\|\tilde{H}_i\|_p^p}{n_i} = \sum_{i=1}^{m} \frac{1}{n_i} \sum_{j \in L_i^-} \left(\tilde{H}_i(x_j^-)\right)^p \qquad (3.7)\\ &= \sum_{i=1}^{m} \frac{1}{n_i} \sum_{j \in L_i^-} \left(\sum_{k \in L_i^+} \log\left(1 + \exp(-\mathbf{u}_i^T(\mathbf{v}_k - \mathbf{v}_j))\right) \right)^p .\end{aligned}$$

The optimization is done by alternating minimization [83], i.e., updating U while keeping V fixed, and then updating V while keeping U fixed. The objective function is bi-convex, i.e., convex in one argument while keeping the other fixed. The updates of the latent factors for each user and each item can be done using gradient descent. In each iteration $t+1$:

$$\mathbf{u}_i^{t+1} \leftarrow \mathbf{u}_i^t - \eta \nabla_{\mathbf{u}_i} \mathcal{E}_{\text{p-push}}(U^t, V^t) \ , \quad i = 1, \ldots, m, \qquad (3.8)$$

$$\mathbf{v}_h^{t+1} \leftarrow \mathbf{v}_h^t - \eta \nabla_{\mathbf{v}_h} \mathcal{E}_{\text{p-push}}(U^{t+1}, V^t) \ , \quad h = 1, \ldots, n. \qquad (3.9)$$

The gradients can be obtained by a direct application of chain rule. For compact notation, let

$$\sigma(x) = \frac{1}{1 + \exp(x)} \ . \qquad (3.10)$$

From (3.3) and (3.4), we get the gradients

$$\nabla_{\Delta_i(k,j)} \ell(\Delta_i(k,j)) = -\sigma(\Delta_i(k,j)) \ , \qquad (3.11)$$

$$\nabla_{\mathbf{u}_i} \Delta_i(k,j) = (\mathbf{v}_k - \mathbf{v}_j) \ , \qquad (3.12)$$

$$\nabla_{\mathbf{v}_h} \Delta_i(k,j) = \begin{cases} \mathbf{u}_i \ , & \text{if } \ h = k \ , \\ -\mathbf{u}_i \ , & \text{if } \ h = j \ . \end{cases} \qquad (3.13)$$

By chain rule, the gradient of the objective w.r.t. $\mathbf{u}_i$ is

$$\nabla_{\mathbf{u}_i} \mathcal{E}_{\text{p-push}}(U,V) = \frac{p}{n_i} \sum_{j \in L_i^-} \left\{ (\tilde{H}_i(x_j^-))^{p-1} \sum_{k \in L_i^+} \sigma(\Delta_i(k,j))(\mathbf{v}_j - \mathbf{v}_k) \right\} .$$

The latent factors $\mathbf{v}_h$ play different roles for different users, depending on whether x_h is relevant or non-relevant for user ρ_i. For item x_h, let Γ_h^+ be the set of users for which x_h is relevant, and let Γ_h^- be the set of users for which x_h is non-relevant. There may be additional users who have no (announced) preference on item x_h, and the optimization of $\mathbf{v}_h$ will not depend on these users. Then, by dividing the sum over all users into sums over Γ_h^+ and Γ_h^- and noting the gradients of $\Delta_i(k,j)$ w.r.t. $\mathbf{v}_h$ for the two cases from (3.13), by chain rule we have

$$\begin{aligned} \nabla_{\mathbf{v}_h} \mathcal{E}_{\text{p-push}}(U,V) = &\sum_{i \in \Gamma_h^-} \frac{p}{n_i} \sum_{j \in L_i^-} \left\{ (\tilde{H}_i(x_j^-))^{p-1} \sum_{k \in L_i^+} \sigma(\Delta_i(k,j)) \mathbf{u}_i \right\} \\ &- \sum_{i \in \Gamma_h^+} \frac{p}{n_i} \sum_{j \in L_i^-} \left\{ (\tilde{H}_i(x_j^-))^{p-1} \sum_{k \in L_i^+} \sigma(\Delta_i(k,j)) \mathbf{u}_i \right\} . \end{aligned}$$

In practice, instead of summing over all users in Γ_h^+, Γ_h^-, one can pick a mini-batch of users to do the update on, in the flavor of stochastic or mini-batch gradient descent.

3.2.3 Infinite Push Collaborative Ranking

The main idea in infinite push collaborative ranking (`Inf-Push CR`), building on [10], is to consider the L_∞ norm, i.e., the largest element, of the surrogate height vector $\tilde{H}_i$, and construct an objective function as a suitable scaled sum of such norms over all users. We consider the objective

$$\begin{aligned} \mathcal{E}_{\text{inf-push}}(U,V) &= \sum_{i=1}^{m} \frac{\|\tilde{H}_i\|_\infty}{n_i} = \sum_{i=1}^{m} \frac{1}{n_i} \max_{j \in L_i^-} \tilde{H}_i(x_j^-) \qquad (3.14) \\ &= \sum_{i=1}^{m} \frac{1}{n_i} \max_{j \in L_i^-} \left(\sum_{k \in L_i^+} \log\left(1 + \exp(-\mathbf{u}_i^T(\mathbf{v}_k - \mathbf{v}_j))\right) \right) . \end{aligned}$$

From an optimization perspective, the infinite push objective above is non-smooth, since it considers the maximum over all elements in $\tilde{H}_i$. One can consider a sub-gradient descent approach, or add auxiliary variables to handle the non-smoothness, and possibly consider the dual problem. Such approaches, however, can be slow in practice, and ignore the specific type of non-smoothness in the problem. In our approach we observe that fixing any one variable U or V, the problem is a smooth minimax problem in the other variable, for which efficient gradient-like first order methods exist [124]. We discuss such efficient updates for U, while keeping V fixed, noting that similar updates are possible for V, leading to suitable alternating minimization algorithms.

Consider the latent factor $\mathbf{u}_i$ corresponding to user ρ_i. Assuming V is fixed, the objective function to minimize for $\mathbf{u}_i$ is given by the max-type function

$$f(\mathbf{u}_i) = \max_{j \in L_i^-} f_j(\mathbf{u}_i) \tag{3.15}$$

where

$$f_j(\mathbf{u}_i) \triangleq \sum_{k \in L_i^+} \log\left(1 + \exp(-\mathbf{u}_i^T \mathbf{a}_{kj})\right) \ , \tag{3.16}$$

with $\mathbf{a}_{kj} = (\mathbf{v}_k - \mathbf{v}_j)$ being a fixed vector. Note that $f_j(\mathbf{u}_i) = \tilde{H}_i(x_j^-)$, and we are using a more suitable notation for the current discussion illustrating the dependency on $\mathbf{u}_i$. For the development, we assume that each $f_j(\mathbf{u}_i)$ are α-strongly convex and β-smooth [124]. For the current setting, the assumptions are satisfied as long as $\mathbf{u}_i, \mathbf{v}_j$ are bounded.

Following [124], we define the linearization of the max-type function $f(\mathbf{u}_i)$ at $\bar{\mathbf{u}}_i$ as

$$f(\bar{\mathbf{u}}_i; \mathbf{u}_i) = \max_{j \in L_i^-} \left[f_j(\bar{\mathbf{u}}_i) + \langle \nabla f_j(\bar{\mathbf{u}}_i), \mathbf{u}_i - \bar{\mathbf{u}}_i \rangle\right] \ . \tag{3.17}$$

To define a gradient mapping, we consider a proximal operator of the linearization, given by

$$f_\gamma(\bar{\mathbf{u}}_i; \mathbf{u}_i) = f(\bar{\mathbf{u}}_i; \mathbf{u}_i) + \frac{\gamma}{2}\|\mathbf{u}_i - \bar{\mathbf{u}}_i\|^2 \ , \tag{3.18}$$

where $\gamma > 0$. Based on the proximal operator, let

$$f^*(\bar{\mathbf{u}}_i;\gamma) = \min_{\mathbf{u}_i} \; f_\gamma(\bar{\mathbf{u}}_i;\mathbf{u}_i) \;, \tag{3.19}$$

$$\mathbf{u}_i^*(\bar{\mathbf{u}}_i;\gamma) = \operatorname*{argmin}_{\mathbf{u}_i} \; f_\gamma(\bar{\mathbf{u}}_i;\mathbf{u}_i) \;, \tag{3.20}$$

$$\mathbf{g}_f(\bar{\mathbf{u}}_i;\gamma) = \gamma(\bar{\mathbf{u}}_i - \mathbf{u}_i^*(\bar{\mathbf{u}}_i;\gamma)) \;, \tag{3.21}$$

where $\mathbf{g}_f(\bar{\mathbf{u}}_i;\gamma)$ is the gradient mapping of the max-type function f at $\bar{\mathbf{u}}_i$ [124, Section 2.3]. Assuming the individual functions f_j, the gradient method for the minimax problem is given by the simple iterative update:

$$\mathbf{u}_i^{t+1} \leftarrow \mathbf{u}_i^t - \eta \mathbf{g}_f(\mathbf{u}_i^t;\gamma) \;, \tag{3.22}$$

where η is the learning rate and $\mathbf{u}_i^t$ serves as the linearization point $\bar{\mathbf{u}}_i$. Since each function $f_j(\mathbf{u}_i)$ is β-smooth, we can set $\gamma = \beta$. With any $\eta \leq \frac{1}{\beta}$, the gradient method has an exponential convergence rate [124, Theorem 2.3.4], which is substantially better than just treating the problem as non-smooth optimization and using sub-gradient descent [124].

The key step in obtaining the gradient mapping in (3.21) is to solve the optimization problem in (3.19). Using the definition of the linearization and the proximal operator [132], we get the following optimization problem:

$$\min_{\mathbf{u}_i} \; \left\{ \max_{j \in L_i^-} [f_j(\bar{\mathbf{u}}_i) + \langle \nabla f_j(\bar{\mathbf{u}}_i), \mathbf{u}_i - \bar{\mathbf{u}}_i \rangle] + \frac{\gamma}{2}\|\mathbf{u}_i - \bar{\mathbf{u}}_i\|^2 \right\} \;. \tag{3.23}$$

The above problem is the proximal operator of a piecewise linear function. The problem can be rewritten as a constrained optimization problem:

$$\begin{aligned} &\min_{\mathbf{u}_i, t} \; t + \frac{\gamma}{2}\|\mathbf{u}_i - \bar{\mathbf{u}}_i\|^2 \\ &\text{s.t.} \quad f_j(\bar{\mathbf{u}}_i) + \langle \nabla f_j(\bar{\mathbf{u}}_i), \mathbf{u}_i - \bar{\mathbf{u}}_i \rangle \leq t \;. \end{aligned} \tag{3.24}$$

A direct calculation shows that the dual of the constrained problem is equivalent to a quadratic minimization problem over the probability simplex:

$$\min_{\mathbf{y}} \mathbf{y}^T Q \mathbf{y} + \mathbf{b}^T \mathbf{y} \quad \text{s.t.} \quad \sum_j y_j = 1, \quad y_j \geq 0 \;, \tag{3.25}$$

where Q is a positive semi-definite matrix. The dual objective is concave, and simply the negative of the above quadratic form, and $\mathbf{y}$ corresponds to the Lagrange multipliers. The quadratic problem with simplex constraints can be solved by a projected gradient or exponentiated gradient method [61, 92]. In practice, we use early stopping of the dual projected gradient descent, compute the gradient mapping, and update $\mathbf{u}_i$ based on (3.22).

3.2.4 Reverse-Height Push Collaborative Ranking

We now consider a formulation based on the reverse height $R_i(x_k^+)$ of positive items x_k^+ for user ρ_i as in (3.2). As discussed in [146], the reverse height is similar to the rank of a positive item x_k^+, but only considering the negative items x_j^- which are above it, and disregarding the other positive items. By treating reverse height as a surrogate for rank, one can follow the existing literature on learning to rank, and consider maximizing objectives such as reciprocal rank or discounted cumulative gain (DCG), e.g., for each user ρ_i, maximize

$$\sum_{k \in L_i^+} \frac{1}{R_i(x_k^+)} \quad \text{or} \quad \sum_{k \in L_i^+} \frac{1}{\log(1 + R_i(x_k^+))} \ .$$

However, these objectives are difficult to directly maximize. Instead, staying in the spirit of these objectives, we consider the following objective

$$\sum_{k \in L_i^+} \log \left(\frac{1}{1 + R_i(x_k^+)} \right) \ ,$$

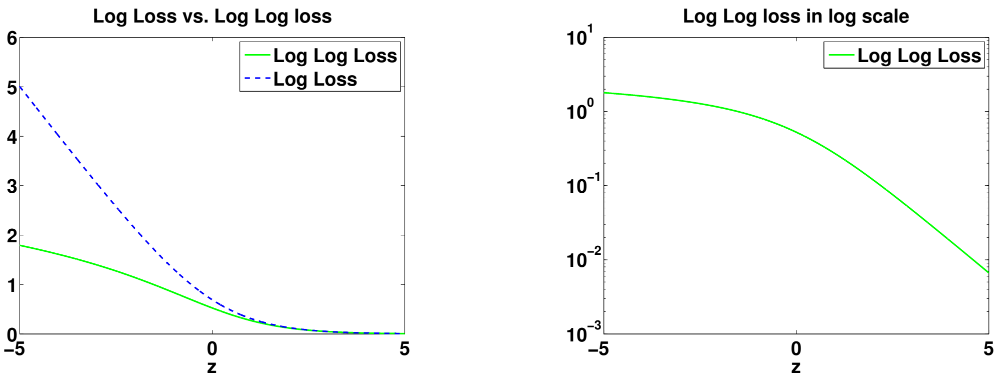
which decreases with increasing rank.

To maintain consistency with the earlier formulations, we consider minimizing the negative of the objective. Further, using the surrogate reverse height $\tilde{R}_i(x_k^+)$ as in (3.6), and considering the objective over all users, we focus on minimizing the following objective:

$$\mathcal{E}_{\text{rh-push}}(U, V) = \sum_{i=1}^{m} \frac{1}{n_i} \sum_{k \in L_i^+} \log(1 + \tilde{R}_i(x_k^+)) \tag{3.26}$$

where

$$\tilde{R}_i(x_k^+) = \sum_{j \in L_i^-} \log\left(1 + \exp(-\mathbf{u}_i^T(\mathbf{v}_k - \mathbf{v}_j))\right) . \tag{3.27}$$



(a) Log-log vs. logistic loss (b) Log-log loss in log-scale

Figure 3.1: (a) A comparison of log-log loss with the logistic loss, and (b) the log-log loss with a y-log scale.

Denoting $\Delta_i(k, j) = z$ for simplicity, the above loss has components of the form $\log(1 + \log(1 + \exp(-z))$, which we refer to as log-log loss. Interestingly, while the log-log loss is not convex, it is quasi-convex and monotonic decreasing, as shown in Figure 3.1. Further, the objective is more robust than the logistic loss for negative z, and stays closer to the 0-1 loss for which it acts as a surrogate. Related non-convex losses have been explored in the literature, e.g., in the context of t-logistic regression [59].

We again use alternating minimization of the objective in (3.26), i.e., updating U keeping V fixed, and updating V keeping U fixed. In the current setting, the alternating minimization will reach a stationary point, possibly a local minimum of the objective. As we discuss in Section 3.3, the empirical results from the log-log loss are surprisingly good along with fairly stable parameter sensitivity, which implies the need for further study of the loss. For alternating minimization, the updates can be done using gradient descent:

$$\mathbf{u}_i^{t+1} \leftarrow \mathbf{u}_i^t - \eta \nabla_{\mathbf{u}_i} \mathcal{E}_{\text{rh-push}}(U^t, V^t) \ , \quad i = 1, \dots, m, \tag{3.28}$$

$$\mathbf{v}_h^{t+1} \leftarrow \mathbf{v}_h^t - \eta \nabla_{\mathbf{v}_h} \mathcal{E}_{\text{rh-push}}(U^{t+1}, V^t) \ , \quad h = 1, \dots, n. \tag{3.29}$$

As before, the gradients can be obtained by chain rule. Using (3.11) and (3.12), from

(3.26) we have

$$\nabla_{\mathbf{u}_i}\mathcal{E}_{\text{rh-push}}(U,V) = \frac{1}{n_i}\sum_{k\in L_i^+}\left\{\frac{1}{1+\tilde{R}_i(x_k^+)}\sum_{j\in L_i^-}\sigma(\Delta_i(k,j))(\mathbf{v}_j-\mathbf{v}_k)\right\} .$$

Recall that the latent factors $\mathbf{v}_h$ play different roles for different users, depending on whether x_h is relevant or not for user ρ_i. As before, for item x_h, let Γ_h^+ be the set of users for which x_h is relevant, and let Γ_h^- be the set of users for which x_h is non-relevant. There may be additional users who have no (announced) preference on item x_h, and the optimization of $\mathbf{v}_h$ will not depend on these users. Then, by dividing the sum over all users into sums over Γ_h^+ and Γ_h^- and noting the gradients of $\Delta_i(k,j)$ w.r.t. $\mathbf{v}_h$ for the two cases from (3.13), from (3.26), by chain rule we have

$$\begin{aligned}\nabla_{\mathbf{v}_h}\mathcal{E}_{\text{rh-push}}(U,V) = &\sum_{i\in\Gamma_h^-}\frac{1}{n_i}\sum_{j\in L_i^-}\left\{\frac{1}{1+\tilde{R}_i(x_k^+)}\sum_{k\in L_i^+}\sigma(\Delta_i(k,j))\mathbf{u}_i\right\}\\ &-\sum_{i\in\Gamma_h^+}\frac{1}{n_i}\sum_{j\in L_i^-}\left\{\frac{1}{1+\tilde{R}_i(x_k^+)}\sum_{k\in L_i^+}\sigma(\Delta_i(k,j))\mathbf{u}_i\right\} .\end{aligned}$$

Again, in practice, instead of summing over all users in Γ_h^+, Γ_h^-, one can pick a mini-batch of users to do the update on, in the flavor of stochastic or mini-batch gradient descent.

For all of our proposed formulations we consider an additional regularization term $\frac{\lambda}{2}(\|U\|^2+\|V\|^2)$. The gradient term for the $\mathbf{u}_i$ updates then has an additional term $\lambda\mathbf{u}_i$, and similarly for the $\mathbf{v}_h$ updates. Also, the computational complexity for computing the gradients for a user ρ_i in one iteration is $\mathcal{O}(n_i^+ n_i^-)$, which is small in practice since $n_i^+, n_i^- \ll n$, the total number of items.

3.3 Experiments

We conducted extensive experiments in order to validate the performance of the three proposed collaborative ranking approaches (`P-Push CR`, `Inf-Push CR`, `RH-Push CR`) on real world datasets. We have divided our experiments into three sets in order to address the following questions:

1. What is the impact of posing the problem as a rating prediction problem versus a ranking one, i.e., optimizing for the square loss versus a ranking loss with focus

Dataset	# Users	# Items	# Ratings
MovieLens 100K	943	1,682	100,000
MovieLens 1M	6,040	3,706	1,000,209
EachMovie	30,000	1,623	2,109,780
Yahoo! R1	6,000	1,000	3,522,232

Table 3.1: Dataset Statistics.

on the top of the list?

2. Is learning the ranking functions *collaboratively* across all users more effective than constructing a separate ranking model per user?

3. Do the proposed push collaborative ranking approaches outperform state-of-the-art CR approaches for recommendation systems?

After discussing the experimental setup, we evaluate each one of these questions empirically.

Datasets. We used four publicly available datasets (Section 2.6): three movie datasets MovieLens 100K, MovieLens 1M, EachMovie and the musical artist dataset from Yahoo!. The users, items and ratings statistics are summarized in Table 3.1.

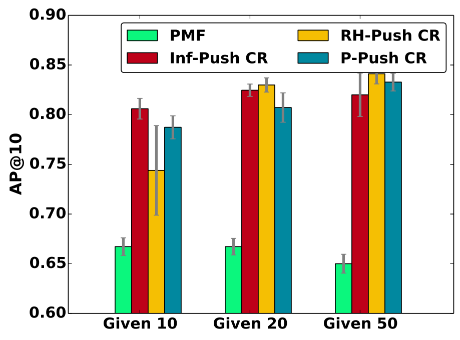
Data Split. Throughout our experiments we adopted the "weak" generalization setting which has become standard practice in the literature [20, 100, 170, 175]. In this setting, we fix the number of training ratings per user $n_i = \{10, 20, 50\}$ and randomly choose n_i ratings for each user for training, 10 ratings for validation, and test on the remaining ratings. The experimental setting facilitates the comparison with the state-of-the-art approaches and allows the study of the ranking algorithms' sensitivity to the number of available training ratings per user. Users with less than $n_i + 20$ ratings are removed to ensure that we can evaluate on at least 10 ratings for each user. Note that the number of test items vary significantly across users, with many users having many more test ratings than training ones, thus simulating the real life recommendation scenario [170]. For each setting of 'given n_i' ratings per user we repeated the whole procedure 10 times and reported the mean and standard deviation of the results.

Setup. In the evaluation phase, for each user, we compare pairs of items $(x_h, x_{h'})$ belonging to the test set for that user, such that each of these items have appeared in the training set of some users. Using such items ensures that the latent factors of the items have been updated during training, and we are not working with their randomly initialized value. In addition, in order to simulate the setup of two-level relevance ratings (Section 3.2), we transformed the ratings of all datasets from a multi-level relevance scale to a two-level scale $(+1, -1)$, i.e., using 4 as relevance threshold for MovieLens 100K, MovieLens 1M and Yahoo!, and 5 for EachMovie. In particular, for the two MovieLens datasets and the Yahoo! dataset, ratings of the value 4-5 are marked as relevant, and 1-3 are marked as non-relevant. For the EachMovie dataset 5-6 are marked as relevant, and 1-4 are marked as non-relevant. Also, we removed users with no positive (relevant) or no negative (non-relevant) ratings in the training set, so that there is at least one positive example to "pull up" or at least one negative example to "push down". The exact same input was given to all algorithms.

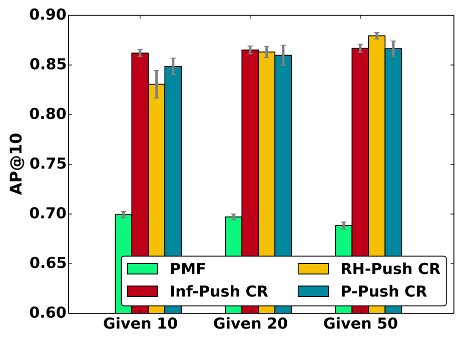
Metrics. Since our goal is to achieve accuracy towards the top of the ranked list, we chose two evaluation metrics which reflect this and have been widely used in learning to rank, *Average Precision@top*, and *NDCG@top* (Section 2.4) [29, 37, 99]. For the definition of *Average Precision@top*, $r_{i[\ell]} = 0$ if this is a non-relevant item for the user, and 1 otherwise. For the computation of *NDCG@top*, $r_{i[\ell]}$ equals to the user's observed rating from the multi-level scale, i.e., 1-6 for EachMovie, 1-5 for the rest of the datasets. The latter choice was made for the sake of consistency with the state-of-the-art literature [170, 175], though during training all approaches have access just to the two-level relevance scale ratings.

In our experiments, we reported the results for MAP@*top* and Mean NDCG@*top* over all users. We varied *top*, also denoted as k in our figures, from 1 to 10 as this is usually the size of a recommendation list fitting a device's screen.

Model Parameters. The validation set was used for selecting the best model parameters. We varied the regularization coefficient λ in the range $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$ and the rank of the latent factors in $\{5, 10, 20, 30, 50, 70, 90\}$. We fixed the learning rate for the gradient based updates to be $\eta = 10^{-4}$ for MovieLens 1M and Yahoo!, and $\eta = 10^{-3}$ for MovieLens 100K and EachMovie. In addition, the validation set was used to determine early stopping. We stopped either after 200 iterations or



(a) MovieLens 100K



(b) MovieLens 1M

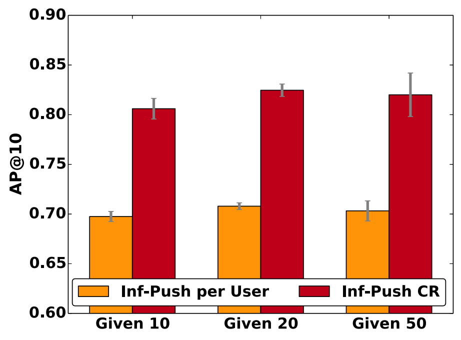
Figure 3.2: Ranking vs. Rating Prediction (`PMF`) for the two MovieLens datasets. All `Push CR` methods outperform `PMF` in terms of AP@10.

when the improvement in MAP@5 in the validation set got smaller than some threshold (10^{-4}).

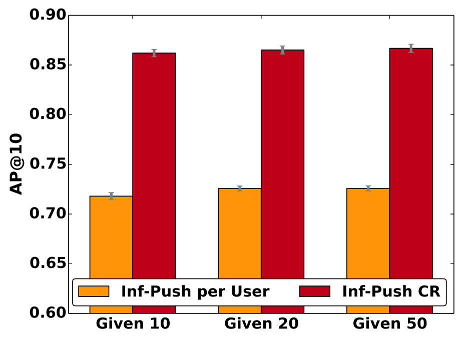
For the `p-Push CR` method we set the parameter $p = 2$, which gives a small push towards the top of the list. The reason we chose a small p was because of numerical instabilities which appeared for higher values of p. For the inner optimization problem (3.23) of the `Inf-Push CR` approach, we used the following parameter setting: We set γ of the proximal operator equal to 10, learning rate equal to 0.01 and regularization coefficient $1/\gamma = 0.1$. Early stopping was used, since there was no need to solve the inner problem fully as in the next outer iteration the same optimization problem would have to be solved. We stopped either when the improvement in the value of the optimization problem was smaller than 0.01 or after 25 iterations.

3.3.1 Validation—Ranking vs. Rating Prediction

In the first set of experiments, we validate the effect of optimizing for a ranking based loss function versus the square loss. Since such a comparison has been explored in the recent literature [20, 175], we limit the set of experiments to a direct comparison between probabilistic matrix factorization (`PMF`) [118] (*rating prediction*) and the proposed `Push CR` approaches (*ranking*) in terms of AP@10. For the experiments, we used the two MovieLens datasets, with the number n_i of training ratings per user set to vary



(a) MovieLens 100K



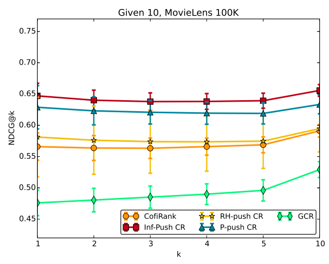
(b) MovieLens 1M

Figure 3.3: `Inf-Push CR` vs. Infinite Push per User for the two MovieLens datasets. Collaboratively learning ranking function results in better top 10 performance than constructing different ranking model per user.

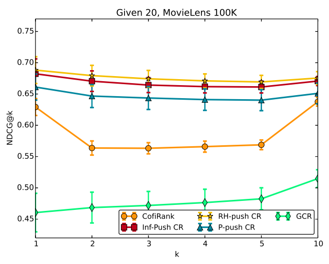
in the range of $\{10, 20, 50\}$. Figures 3.2(a) and 3.2(b) illustrate the superiority of optimizing a ranking based loss function compared to the square loss. The results support the effectiveness of posing recommendation as a ranking problem rather than a rating prediction problem based on squared loss. Thus, in the rest we focus on comparing ranking approaches.

3.3.2 Validation—Collaborative vs. Individual

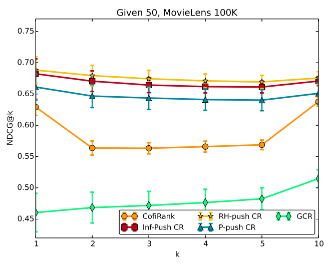
The second set of experiments is conducted to examine whether collaborative ranking, i.e., learning latent factors for the users and items from the rating matrix, results in higher ranking accuracy compared to training separate ranking functions per user. For this task, we compare the proposed collaborative model for Infinite Push (`Inf-Push CR`) with the approach of building a separate Infinite Push model per user (Inf-Push per User). We again used the two MovieLens datasets, varying the number n_i of training ratings per user in the range of $\{10, 20, 50\}$ and reported the mean and standard deviation of AP@10. For Inf-Push per User we implemented Infinite Push as in [10] and used the feature vectors of the items (movie genre) in order to learn the linear ranking function [5, 156]. Note that according to [156], genre seems to be the most informative feature among other types of side information. Further, any helpful feature one can add for the items, the collaborative model can also be extended to take advantage of such



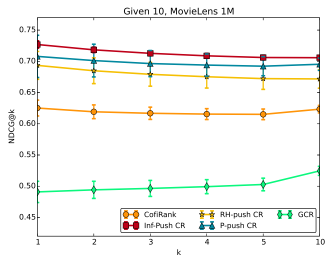
(a) MovieLens 100K, Given10



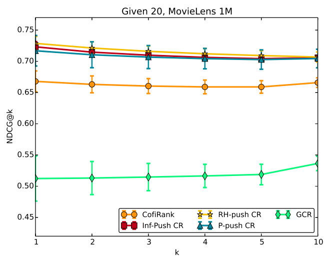
(b) MovieLens 100K, Given20



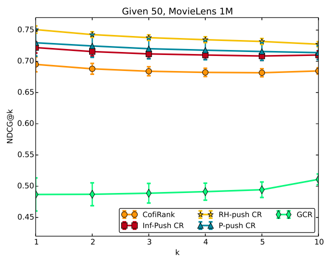
(c) MovieLens 100K, Given50



(d) MovieLens 1M, Given10



(e) MovieLens 1M, Given20



(f) MovieLens 1M, Given50

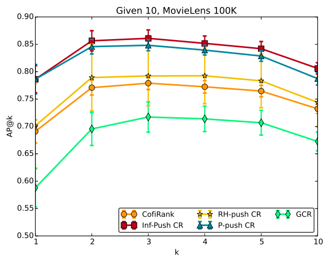
Figure 3.4: Comparison of performance of the proposed `Push CR` models against `CofiRank` and `GCR` in NDCG@*top* for $top = \{1, 2, 3, 4, 5, 10\}$ for the two MovieLens datasets. All of our proposed `Push CR` methods outperform `CofiRank` and `GCR`.

features. Figures 3.3(a) and 3.3(b) show that learning the latent factors collaboratively clearly outperforms learning a separate ranking function per user using the rated items' features. The experiment validates the effectiveness of the collaborative ranking framework and allows us to focus on CR approaches for recommendation for the rest of this chapter.

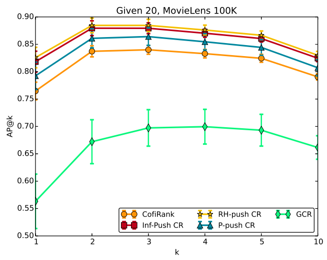
3.3.3 Performance of Proposed Approaches

In the third set of experiments, we address the question: Do our three proposed CR approaches, i.e., `p-Push CR`, `Inf-Push CR`, `RH-Push CR` outperform the state-of-the-art CR methods? We compare the `Push CR` methods with `CofiRank` [175], which is considered a very popular baseline in the CR literature, and with Global Collaborative Ranking (`GCR`) [100] which is the ranking counterpart of PMF-type methods. For `CofiRank`, we used the publicly available package[1] and set the same parameter values provided in the

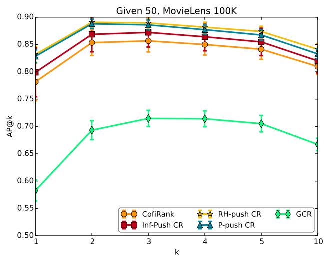
[1] www.cofirank.org



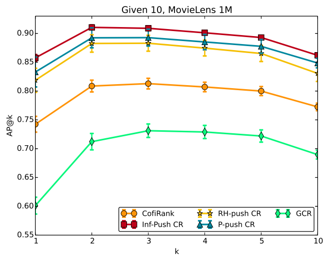
(a) MovieLens 100K, Given10



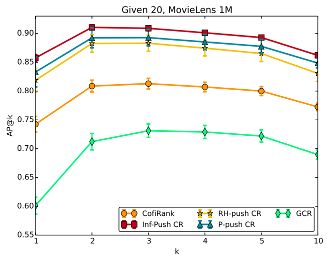
(b) MovieLens 100K, Given20



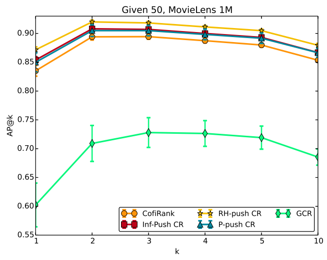
(c) MovieLens 100K, Given50



(d) MovieLens 1M, Given10



(e) MovieLens 1M, Given20



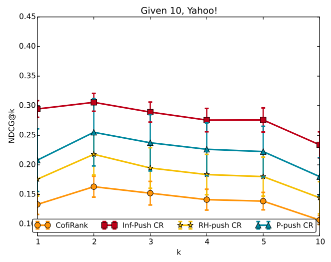
(f) MovieLens 1M, Given50

Figure 3.5: Comparison of performance of the proposed `Push CR` models against `CofiRank` and `GCR` in Average Precision@*top* for *top* $k = \{1, 2, 3, 4, 5, 10\}$ for the two MovieLens datasets. All of our proposed `Push CR` methods outperform `CofiRank` and `GCR`.

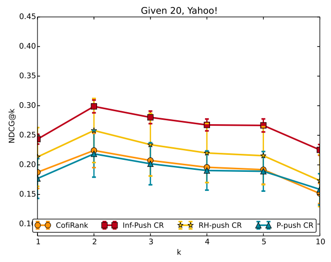
original paper [175] (rank = 100 and $\lambda = 10$), and default values provided in the source code for unstated parameters. For `GCR`, we implemented the source code based on the publicly available package[2] since the original code used slightly different experimental setting than ours, making the comparison of the different methods difficult. In the `GCR` experiments, we chose the model rank in the validation set, varying it in the range $\{5, 10, 15, 20\}$ and the regularization parameter λ in $\{10^{-3}, 10^{-2}, 10^{-1}\}$, and kept the rest of the parameters in their default values given in the package. Also, we optimized for the average number of mis-ordered pairs (0-1 error) [100, Eq (6)] and we stopped the algorithm either when the improvement in the 0-1 error in the validation set got smaller than some threshold (0.001) or after 200 iterations.

Results and Discussion. We report the mean and standard deviation of AP@top and NDCG@top over 10 runs, where *top k* varies in the range of {1, 2, 3, 4, 5, 10}

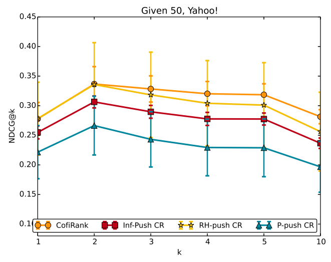
[2] prea.gatech.edu



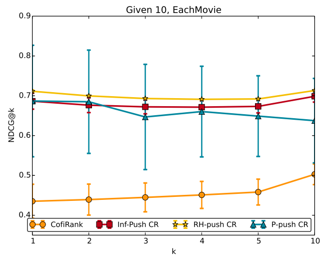
(a) Yahoo!, Given10



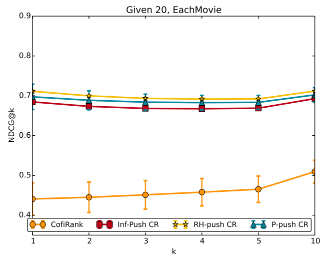
(b) Yahoo!, Given20



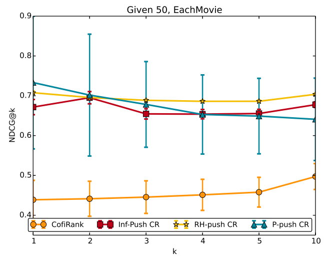
(c) Yahoo!, Given50



(d) EachMovie, Given10



(e) EachMovie, Given20

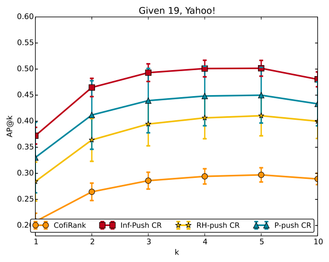


(f) EachMovie, Given50

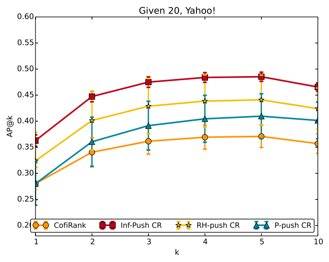
Figure 3.6: Comparison of performance of the proposed `Push CR` models vs. `CofiRank` in NDCG@*top* for the Yahoo! and EachMovie datasets. In most cases at least one (usually all) of our proposed `Push CR` methods outperforms `CofiRank`.

and the fixed number of training ratings per user (n_i) varies in {10, 20, 50}. Recall that the parameters, such as rank, regularization parameter, and early stopping, for the Push CR methods were all chosen based on performance on AP@5 in the validation set. Figures 3.4(a)-(c), 3.4(d)-(f), 3.6(a)-(c), 3.6(d)-(f) compare the competing methods in terms of NDCG@top for the MovieLens 100K, 1M, EachMovie and Yahoo! datasets respectively, whereas figures 3.5(a)-(c), 3.5(d)-(f), 3.7(a)-(c), 3.7(d)-(f) show the AP@top comparison.

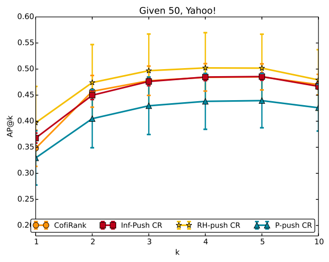
Based on the experimental results illustrated in Figures 3.4, 3.5, 3.6, 3.7 we conclude that in most cases at least one (usually all) of our proposed `Push CR` methods outperforms `CofiRank` in terms of both ranking metrics. The only case where the mean of NDCG@top of `CofiRank` outperforms the proposed approaches is for Yahoo!, Given 50. For the same case, the mean of AP@top of `CofiRank` almost coincides with that of `Inf-Push CR`, is higher than that of `P-Push CR` but is outperformed by `RH-Push CR`. For the case of Yahoo!, Given 20, the NDCG@ top of CofiRank outperforms that of



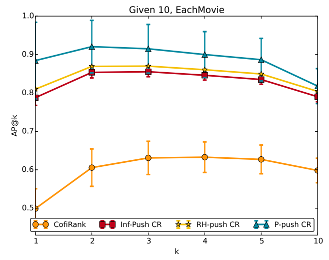
(a) Yahoo!, Given10



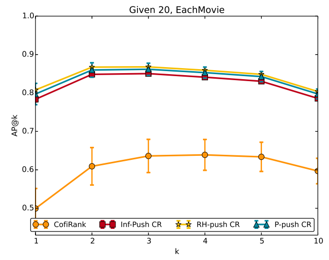
(b) Yahoo!, Given20



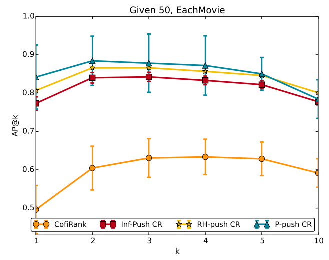
(c) Yahoo!, Given50



(d) EachMovie, Given10



(e) EachMovie, Given20



(f) EachMovie, Given50

Figure 3.7: Comparison of performance of the proposed `Push CR` models vs. `CofiRank` in AP@*top* for the Yahoo! and EachMovie datasets. In all cases at least one (usually all) of our proposed `Push CR` methods outperforms `CofiRank`.

`P-Push CR` but lags behind `Inf-Push CR` and `RH-Push CR`. So, overall, our `Push CR` approaches significantly outperform `CofiRank`. In addition, from our experiments in the two MovieLens datasets, we observe that `CofiRank` and all proposed `Push CR` methods always outperform `GCR` in terms of the two top-ranking metrics.

We can also compare the performance among the proposed `Push CR` methods. The improvement of `Inf-Push CR` is particularly notable when the amount of available training points (n_i) is small. For n_i=10, `Inf-Push CR` outperforms all other methods in terms of both AP@top and NDCG@top, in all datasets apart from EachMovie (where it lags behind `P-Push CR`). In contrast, the improvement of `RH-Push CR` is particularly notable when the amount of available training points (n_i) grows larger. For n_i=50, `RH-Push CR` outperforms the rest of Push CR methods in both AP@top and NDCG@top, except for EachMovie, where the mean of AP@top lags behind that of `P-Push CR`. Also, `P-Push CR` has the most variability in its performance across the ten runs.

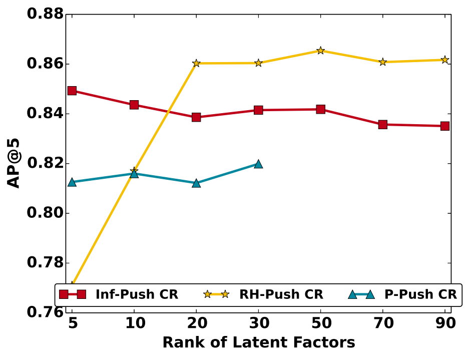
Comparison with CLiMF. For completeness, we also compare the proposed `Push CR` methods with another state-of-the-art method, `CLiMF` [159]. Such a comparison is tricky as `CLiMF` uses implicit binary ratings data while `Push CR` methods use explicit binary ratings. We train `CLiMF` on the explicit feedback datasets by binarizing the rating values using 4 as the relevance threshold, treating ratings of value 1-3 as 0 (not observed). Otherwise, we evaluate `CLiMF` using exactly the same setup as in the rest of the methods. For this experiment, we use the MovieLens 100K and 1M datasets, fix the number of training ratings per user $n_i = 20$, choose best parameters (λ: $\{10^{-4}$, 10^{-3}, 10^{-2}, 10^{-1},$\}$, rank: $\{5, 10, 20, 30, 50, 70, 90\}$) in the validation set and report the mean of AP@5 and NDCG@5 over ten random runs. Results are reported in Table 3.2 and show that the `Push CR` methods are competitive, usually better on AP@5 and NDCG@5 compared to CLiMF. Similar trends were observed for different values of 'top' and n_i, and are thus omitted.

	MovieLens 100K		MovieLens 1M	
Method	AP@5	NDCG@5	AP@5	NDCG@5
`CLiMF`	0.8446	0.6424	0.8787	0.6982
`Inf-Push CR`	0.8605	0.6612	**0.8938**	0.7042
`RH-Push CR`	**0.8665**	**0.6693**	0.8926	**0.7092**
`P-Push CR`	0.8443	0.6402	0.8865	0.7028

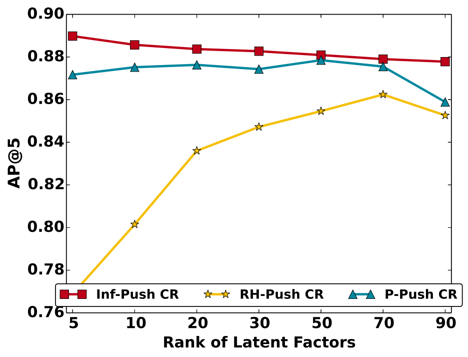
Table 3.2: Comparison of `Push CR` methods with CLiMF.

3.3.4 Parameter Sensitivity Analysis

Figures 3.8 and 3.9 show how the Push CR models behave with different combinations of parameters. We used the two MovieLens datasets, fixing the number of training ratings per user n_i=10. To study the effect of the rank of the latent factors on the model performance, we varied it in the range of $\{5, 10, 20, 30, 50, 70, 90\}$ while keeping the regularizer λ fixed to 0.0001. Figures 3.8(a)-(b) show that there is a decrease in AP@5 for high values of rank (70, 90). For `P-Push CR`, in the MovieLens 100K dataset, higher values of rank resulted in numerical instability issues. Also, in order to study the effect of the regularizer on the model performance, we varied λ in the range of $\{0.0001, 0.001, 0.01, 1\}$ while keeping the rank fixed to 10. Figures 3.9(a)-(b) show that for all three methods there is a decrease in AP@5 for high values of λ (0.1, 1). We



(a) MovieLens 100K



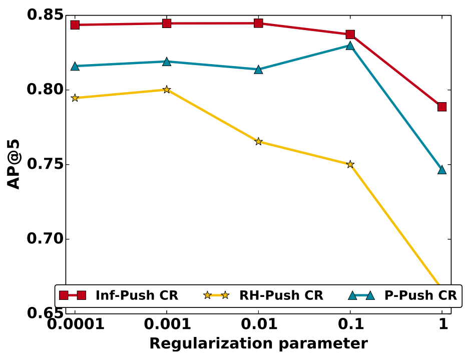
(b) MovieLens 1M

Figure 3.8: Effect of Rank of Latent Factors on AP@5 for MovieLens 100K and MovieLens 1M, for $n_i = 10$. The rank varies in $\{5, 10, 20, 30, 50, 70, 90\}$ while the regularizer is fixed to 0.0001. There is a decrease in AP@5 for high values of rank, e.g., (70, 90).

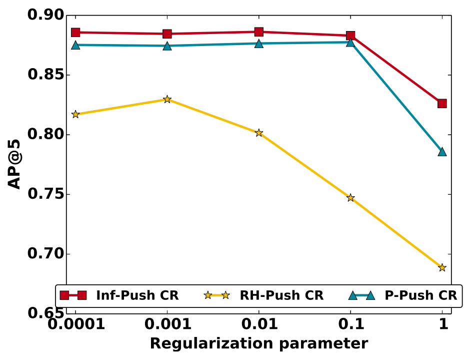
have performed early stopping based on the validation set, leading to robustness of our methods to different parameter choices.

3.4 Related Work

In Information Retrieval (IR), the Learning To Rank (LTR) problem considers a set of training queries, where for each query a set of retrieved documents along with their relevance scores are given [32, 33, 35, 37, 111, 153]. The goal is to learn a ranking function which maps the features of the documents to a relevance score, based on which the documents will be ranked by degree of predicted relevance. A vast literature for LTR exists, and the categories of the methods proposed to approach the problem are *pointwise* [53], *pairwise* [10, 29, 32, 33, 146, 153] and *listwise* [35, 178]. The *pointwise* techniques require fitting to the actual relevance values, which is analogous to optimizing for the AUC metric. Thus, pointwise techniques try to predict the exact relevance scores of unseen documents instead of just capturing the correct ordering between them. The *pairwise* approach avoids these drawbacks by predicting the preference order between data. The two LTR approaches we employ, i.e., P-Norm Push [146] and Infinite Push [10] both lie in the category of pairwise LTR approaches. Their main disadvantage is the computational complexity, which is however not prohibitive when being applied to recommender systems, because of the sparsity of ratings for the users. *Listwise*



(a) MovieLens 100K



(b) MovieLens 1M

Figure 3.9: Effect of regularization parameter on AP@5 for MovieLens 100K and MovieLens 1M, for $n_i = 10$. The regularizer λ varies in $\{0.001, 0.001, 0.1, 1\}$ while rank= 10. For high values of λ, AP@5 decreases.

approaches learn a ranking model for the entire set of documents.

Inspired by the analogy between query-document relations in IR and user-item relations in recommender systems, many CR approaches have been proposed over the past few years for personalized recommendations [20, 95, 100, 141, 159, 157, 158, 170, 175]. The majority of them optimize one of the (top-N) ranking evaluation metrics by exploiting advances in structured estimation [168] that minimize convex upper bounds of the loss functions based on these metrics [99]. `CofiRank` is one of the state-of-the-art listwise CR approaches which optimizes a convex relaxation of the NDCG measure for explicit feedback data (i.e., ratings) [175]. `CLiMF` and xCLIMF respectively optimize a smooth lower bound for the mean reciprocal rank on implicit (based on user behavior) feedback data [159], and expected reciprocal rank for data with multiple levels of relevance [158]. GAPfm [157] has been proposed to optimize Graded Average Precision, in order to address the concerns with Average Precision in multiple-relevance graded data. Recent work in [170], in the same spirit as [20], argued the need to bridge the ideas in the LTR community with Collaborative Filtering (CF). For that, they used a novel approach of generating features for the LTR task using CF techniques, i.e., neighborhood-based and model-based techniques, so that the problem can be transformed to a classic pairwise LTR problem. Recently, [100] introduced a method for Local Collaborative Ranking (LCR) where ideas of local low rank matrix approximation were applied to the pairwise ranking loss minimization framework. A special case of this framework is Global Collaborative Ranking (GCR) where global low rank structure is assumed, as is usually done in the ranking counterpart of Probabilistic Matrix Factorization (PMF)-type methods [5, 148, 156], although equal importance is given to the whole ranked list.

3.5 Summary

This chapter builds on and merges two keys themes from the collaborative filtering and learning to rank literature: the use of a low rank representation for collaborative scoring, and the use of ranking losses which focus on the accuracy at the top of the list, by pushing down non-relevant items and/or pulling up relevant items. The merge results in a family of novel collaborative ranking algorithms with a push at the top of the ranked list for each user. The proposed algorithms are simple and scalable from an optimization perspective and are shown to be competitive to state-of-the-art baselines through extensive experiments.

The modular structure of the development shows that other representations, beyond low rank, and other ranking losses, beyond the three considered here, can be considered with simple changes to the ideas presented. Our methods are applicable in a variety of settings where (i) explicit binary relevance ratings are available (i.e., helpfulness of a review, like/dislike of a song) and (ii) implicit binary ratings are treated as explicit, for example considering non-clicks as non-relevant [57], and using proxies like dwell time as a threshold to quantify item relevance [179]. A natural extension for `Push CR` methods is to handle multi-level relevance ratings, and incorporate feature information from users and items when available.

Chapter 4

Recommendation as a Collective Process

How can we give good product recommendations
that do not result in 'out of stock' messages?

In many recommendation settings, the candidate items for recommendation are associated with a maximum *capacity*, i.e., number of seats in a Point-of-Interest (POI) or number of item copies in the inventory. However, despite the prevalence of the capacity constraint in the recommendation process, the existing recommendation methods are not designed to optimize for respecting such a constraint.

Towards closing this gap, in this chapter we propose *Recommendation with Capacity Constraints*—a framework that optimizes for both recommendation accuracy and expected item usage that respects the capacity constraints. We show how to apply our method to three state-of-the-art latent factor recommendation models: probabilistic matrix factorization (`PMF`), bayesian personalized ranking (`BPR`) for item recommendation, and geographical matrix factorization (`GeoMF`) for POI recommendation. Our experiments indicate that our framework is effective for providing good recommendations while taking the limited resources into consideration. Interestingly, our methods are shown in some cases to further improve the top-N recommendation quality of the respective unconstrained models.

4.1 Introduction

Consider what would happen if a Point-of-Interest (POI) recommendation system suggests to a large number of users to visit the same POI, e.g. the same attraction in a theme park, or the same coffee shop; or what would be the effect of an item recommendation system recommending the same products (e.g. movies, jackets) to the vast majority of customers. It is easy to imagine that in the first case the recommended POI might get overcrowded, resulting in long queues, thus high waiting times. In the second case, the customers might view an 'out of stock' or 'server overload' message. In either scenario, the user experience will be deteriorated.

The above scenarios, although seemingly different, share the following key property: every item candidate for recommendation is associated with a maximum *capacity*—for a POI it could be the number of visitors allowed at the same time or the number of seats or tables; for a product it could be the maximum number of copies that can be purchased/consumed simultaneously.

As recommendation systems become more prevalent and touch more aspects of everyday life, there is an increase in potential applications that require providing recommendations while respecting the capacity constraints for the items. A few interesting examples that can be expressed in this setting are: (1) book recommendation systems employed by libraries, where the books recommended to the borrowers should be on the shelf, (2) route recommendation systems, which aim to suggest the best road for driving while keeping the roads from getting congested, (3) class recommendation systems employed by universities, where every recommended class can accept a limited number of students, (4) viral content recommenders, which serve personalized content in rush watching periods such as Prime Time or the Oscars, and many more.

To the best of our knowledge, none of the state-of-the-art recommendation approaches for item and POI recommendation is designed to respect the capacity constraints. Instead, they are often designed to optimize for rating prediction [148, 156] or personalized ranking accuracy [46, 141, 159], or other metrics such as serendipity [67], novelty [82], and diversity [84].

We propose a novel approach that both optimizes for recommendation accuracy, measured suitably by rating prediction or personalized ranking losses, and penalizes excessive usage of items that surpasses the corresponding capacities. We show how to apply our approach to three state-of-the-art latent factor recommender models: probabilistic matrix factorization (`PMF`) [148], geographical matrix factorization (`GeoMF`) [108], and bayesian personalized ranking (`BPR`) [141]. We introduce the concepts of *user propensity* (i.e., the probability to follow the recommendations) and *item capacity*. Both concepts are key factors for estimating the extent to which the expected usage of the items exceeds the capacities. Our experimental results in real-world recommendation datasets show that our formulation (i) is suitable for different choices of propensities, capacities, and surrogate loss for the capacity objective, and (ii) allows for recommendations which respect the capacity constraints, while the recommendation quality is not deteriorated by a lot; in fact, in some cases our methods outperform the state-of-the-art in top-N recommendation quality.

The rest of the chapter is organized as follows. In Section 4.2 we introduce the concepts of user propensities and item capacities. In Section 4.3 we devise our methods for recommendation with capacity constraints. We empirically evaluate our method in Section 4.4, review related work in Section 4.5, and summarize in Section 4.6.

4.2 Key Concepts and Preliminaries

We start with introducing some useful concepts.

4.2.1 Item Capacities

Every item $j = 1, \ldots, N$ is characterized by a parameter indicating the maximum number of users who can simultaneously use it. We refer to this variable as *capacity* and denote it with $c_j > 0$, resulting in a vector of capacities $\mathbf{c} \in \mathbb{R}_+^N$ for all N items. For POI recommendation, a POI's capacity could be the total number of seats, or number of visitors allowed per time slot. For general item recommendation, an item's capacity could be the maximum number of users that can watch the same movie online without leading to a system crash, or the maximum number of copies of the same item in the inventory.

Capacity is key for recommendation systems, as when many users are directed to the same item, the item will quickly reach its capacity. This will in turn lead to deteriorated user experience, such as long waiting times or out of stock items.

4.2.2 User Propensities

Every user i is associated with a variable, indicating the probability that he will follow the system's recommendations. We refer to this variable as *user propensity* and denote it with $p_i \in [0, 1]$, resulting in a vector of propensities $\mathbf{p} \in \mathbb{R}^M$ for all M users. There are many ways propensities can be modeled; one possible definition is:

$$p_i = \frac{\#\text{ times user } i \text{ followed the recommendation}}{\#\text{ user } i\text{-system interactions}},$$

which uses feedback of the form user follows/ignores the recommendation. Alternative definitions are explored in Section 4.4.

Propensity to follow the recommendations can be an inherent user property: some users tend to listen to the system's recommendations more, compared to others. However, the same user's propensity might vary with time, e.g. the user might go to a certain place for lunch every day at 2pm, so no matter how good the restaurant recommendation is, he will not listen (low propensity); in contrast, he might be experimental during his dinner time (high propensity). Also, factors such as who the user is with or quality of experience with the system can affect the user's propensity.

We argue that user propensity, which has connections to the themes of [109, 151], is a key factor for recommendation systems. Though we use it to compute the expected usage of items, one can also use it to e.g. target the users with low/high propensities.

4.2.3 Geographical Matrix Factorization (GeoMF)

Here we briefly review `GeoMF` as we use it as the basis recommendation model for Point-of-Interest (POI) recommendation.

In POI recommendation, the rating matrix R contains the check-in data of M users on N POIs. Because of the natural characterization of POIs in terms of their geographical location (latitude and longitude) and the spatial clustering in human mobility patterns, recommendations can be further improved [181]. This insight has led researchers to the development of `GeoMF` [108] which jointly models the geographical and MF component. The key idea behind `GeoMF`, and the related works of [107, 181], is that if a user i has visited a POI j but has not visited the nearby POIs, these nearby POIs are more likely to be disliked by this user compared to far-away non-visited POIs. `GeoMF` represents the users and items not only by their latent factors U, V, but also by the activity and influence vectors X, Y respectively. By splitting the entire world into

L even grids, GeoMF models a user i's *activity area* as a vector $\mathbf{x}_i \in \mathbb{R}^L$, where every entry $x_{i\ell}$ of the vector denotes the possibility that this user will appear in the ℓ-th grid. Similarly, the model represents every POI by a vector $\mathbf{y}_j \in \mathbb{R}^L$, referred to as *influence vector*, where every entry $y_{j\ell}$ indicates the quantity of influence POI j has on the ℓ-th grid. While the activity area vector $\mathbf{x}_i$ of every user i is latent, the influence vectors $\mathbf{y}_j$ for every POI j are given as input to the model and are pre-computed using kernel density estimation [182] as follows. The degree of the influence POI j has on the ℓ-th grid is computed as $y_{j\ell} = \frac{1}{\sigma}\mathcal{K}(\frac{d(j,\ell)}{\sigma})$ where $\mathcal{K}$ is the standard Gaussian distribution, σ is the standard deviation and $d(j,\ell)$ is the Euclidean distance between the POI j and the ℓ-th grid. Although in [108] the activity area vectors are constrained to be non-negative and sparse, in our work we do not make such an assumption. Concretely, GeoMF predicts user i's rating on POI j as

$$\hat{r}_{ij} = \mathbf{u}_i^T \mathbf{v}_j + \mathbf{x}_i^T \mathbf{y}_j. \tag{4.1}$$

GeoMF uses the point-wise square loss of PMF; however, we can also replace this with the BPR loss, giving rise to a method we refer to as GeoBPR.

4.3 Recommendation with Capacity Constraints

As discussed in Section 4.1, many recommendation scenarios ranging from viral video recommendation, limited shelf on a virtual store, road congestion, to POI recommendation, can benefit from a system that gives recommendations while respecting the capacity constraints. In these settings, we want good recommendation accuracy, while the items' estimated usage does not exceed the respective capacities. To achieve this, we discuss two alternative approaches.

4.3.1 Approach 1: Post-Processing

The first simple method we discuss is referred to as *post-process*. This method takes as input the predicted scores $\hat{r}_{ij}$ for every user - item (i, j) pair. Although we use the scores provided by either of the models of PMF, BPR, GeoMF, GeoBPR, any recommendation model can be used in place. The idea of this approach is that for every item j with capacity c_j, we find the c_j users with the top predicted scores, i.e., who are the users who want item j the most, up until it reaches its capacity? To provide top-N recommendations per user, keeping track of which users are allocated to which items, we sort the assigned items based on the predicted scores in descending order. Note that by construction, this approach achieves recommendations that respect the capacity constraints.

Algorithm 1 *post-process*

```
Require: $\forall j\ c_j$, $\forall (i,j)\ \ \hat{r}_{ij}$ from unconstrained method
1: For every item $j$, (i) rank users based on $\hat{r}_{ij}$, (ii) recommend item $j$ only to top $c_j$ users.
2: For every user $i$, rank the assigned items from step 1, based on $\hat{r}_{ij}$, and
3: return top-N recommendation list.
```

4.3.2 Approach 2: Proposed Framework

The second approach we introduce is formalizing the problem of recommendation with capacity constraints as a weighted sum between two objectives: (i) we want to both optimize for recommendation accuracy, while (ii) penalizing when the items' expected usage exceeds the respective capacities. To balance the two objectives we use a trade-off parameter $\alpha \in [0, 1]$. We will refer to our proposed approaches under this framework as *weighted objectives.*

Personalized Scoring. To provide personalized recommendations, we follow the representation used by `PMF`/`BPR` for item recommendation, and the one used by `GeoMF`/`GeoBPR` in POI recommendation. Specifically, the predicted score of user i on item j is:

$$\hat{r}_{ij} = \begin{cases} \mathbf{u}_i^T \mathbf{v}_j & \text{for item recommendation} \\ \mathbf{u}_i^T \mathbf{v}_j + \mathbf{x}_i^T \mathbf{y}_j & \text{for POI recommendation} \end{cases} \tag{4.2}$$

Note that any model estimating whether a user i will buy/visit item/POI j can be used to replace (4.2), thus leading to a family of recommendation with capacity constraints algorithms.

Expected Usage. We define the expected usage of an item j as the expected number of users who have been recommended item j and will follow the recommendation: $\sum_{i=1}^{M} p_i \hat{r}_{ij}$. If $\hat{r}_{ij}$ was either 1 or 0, the $\sum_{i=1}^{M} \hat{r}_{ij}$ term would indicate the total number of users who have been recommended item j. For ease of optimization, we do not threshold $\hat{r}_{ij}$ to be either 0 or 1. Instead, we constrain $\hat{r}_{ij}$ in the range of [0, 1] by using the sigmoid function, $\sigma(\cdot) = \frac{1}{1+\exp(-(\cdot))}$:

$$\mathbb{E}[\text{usage}(j)] = \sum_{i=1}^{M} p_i \sigma(\hat{r}_{ij}) \ , \tag{4.3}$$

which is the weighted combination of the estimated ratings for the users, using as weights the corresponding user propensities.

Capacity Loss. Since we wish to penalize the model for giving recommendations which result in the expected usage of the items exceeding the corresponding capacities, we want to minimize the average capacity loss, given by:

$$\frac{1}{N} \sum_{j=1}^{N} \mathbb{1}[c_j \leq \mathbb{E}[\text{usage}(j)]] \ . \tag{4.4}$$

Given that the use of the indicator function $\mathbb{1}[\cdot]$ is not suitable for optimization purposes, we will use a surrogate for the indicator function. Considering the difference

$$\Delta(c_j, \mathbb{E}[\text{usage}(j)]) = c_j - \mathbb{E}[\text{usage}(j)], \tag{4.5}$$

we use the logistic loss of the difference as the surrogate:

$$\ell(\Delta(c_j, \mathbb{E}[\text{usage}(j)])) = \log(1 + \exp(-\Delta(c_j, \mathbb{E}[\text{usage}(j)])) \ , \tag{4.6}$$

noting that it forms a convex upper bound to the indicator function. Alternative surrogate losses to consider are:

$$\ell(\Delta(c_j, \mathbb{E}[\text{usage}(j)])) = \begin{cases} \exp(-\Delta) & \text{(Exponential loss)} \\ \max(-\Delta, 0) & \text{(Hinge Loss)} \end{cases} \tag{4.7}$$

Although the square loss $(-\Delta)^2$ is a convex surrogate of the indicator as well, it is not a suitable surrogate for the capacity loss, as it penalizes both positive and negative differences, whereas we want to penalize only when the expected usage *exceeds* the capacity.

Overall Objective. Depending on which loss to minimize in order to capture recommendation quality, i.e., rating prediction loss or pairwise ranking loss, we devise the following four methods: `Cap-PMF`, `Cap-BPR` for item recommendation, and `Cap-GeoMF`, `Cap-GeoBPR` for POI recommendation.

Putting everything together, using the logistic loss as the surrogate loss, our method minimizes the following objective for item recommendation (`Cap-PMF`):

$$\mathcal{E}_{\texttt{cap-PMF}}(U,V) = (1-\alpha)\cdot\sum_{i=1}^{M}\sum_{j\in L_i}(r_{ij}-\mathbf{u}_i^T\mathbf{v}_j)^2 \tag{4.8}$$
$$+\alpha\cdot\frac{1}{N}\sum_{j=1}^{N}\log\left(1+\exp\left(\sum_{i=1}^{M}p_i\sigma(\mathbf{u}_i^T\mathbf{v}_j)-c_j\right)\right)+\lambda(\|U\|_F^2+\|V\|_F^2)\ ,$$

while for POI recommendation our method `Cap-GeoMF` minimizes:

$$\mathcal{E}_{\texttt{cap-GeoMF}}(U,V,X) = (1-\alpha)\cdot\sum_{i=1}^{M}\sum_{j\in L_i}(r_{ij}-\mathbf{u}_i^T\mathbf{v}_j-\mathbf{x}_i^T\mathbf{y}_j)^2, \tag{4.9}$$
$$+\alpha\cdot\frac{1}{N}\sum_{j=1}^{N}\log\left(1+\exp\left(\sum_{i=1}^{M}p_i(\sigma(\mathbf{u}_i^T\mathbf{v}_j+\mathbf{x}_i^T\mathbf{y}_j))-c_j\right)\right)+\lambda(\|U\|_F^2+\|V\|_F^2+\|X\|_F^2)\ .$$

In the above formulation, α is a fixed, user-chosen parameter in [0, 1] that handles the trade-off between the prediction loss and the *capacity loss*. When α is equal to 0, our model reduces to `PMF` (or `GeoMF`). When α is 1, we are not interested in good prediction accuracy—our only concern is to ensure that every item's expected usage will not exceed its fixed capacity.

If we replace the first objective in (4.8) (or (4.9)) with the one of BPR to optimize for ranking accuracy, we obtain `Cap-BPR` (or `Cap-GeoBPR`). For `Cap-BPR` we optimize for:

$$\mathcal{E}_{\texttt{cap-BPR}}(U,V) = (1-\alpha)\cdot\sum_{i=1}^{M}\sum_{k\in L_{i,+}}\sum_{j\in L_{i,-}}\log(1+\exp(-\mathbf{u}_i^T(\mathbf{v}_k-\mathbf{v}_j))) \tag{4.10}$$
$$+\alpha\cdot\frac{1}{N}\sum_{j=1}^{N}\log\left(1+\exp\left(\sum_{i=1}^{M}p_i\sigma(\mathbf{u}_i^T\mathbf{v}_j)-c_j\right)\right)+\lambda(\|U\|_F^2+\|V\|_F^2).$$

Learning the model. Following, we give the gradient updates only for `Cap-PMF` and `Cap-GeoMF`. The optimization is done by alternating minimization. The latent factor updates are done using gradient descent, and for iteration $t+1$ they are:

$$\begin{aligned}
\forall i = 1, \dots, M, &\quad \mathbf{u}_i^{t+1} \leftarrow \mathbf{u}_i^t - \eta \nabla_{\mathbf{u}_i} \mathcal{E}_{\texttt{capMF}}(U^t, V^t, X^t) \\
\forall j = 1, \dots, N, &\quad \mathbf{v}_j^{t+1} \leftarrow \mathbf{v}_j^t - \eta \nabla_{\mathbf{v}_j} \mathcal{E}_{\texttt{capMF}}(U^{t+1}, V^t, X^t) \\
\forall i = 1, \dots, M, &\quad \mathbf{x}_i^{t+1} \leftarrow \mathbf{x}_i^t - \eta \nabla_{\mathbf{x}_i} \mathcal{E}_{\texttt{cap-GeoMF}}(U^{t+1}, V^{t+1}, X^t)
\end{aligned}$$

where the last gradient update is valid only for `Cap-GeoMF`. We use $\mathcal{E}_{\texttt{CapMF}}$ to denote either $\mathcal{E}_{\texttt{Cap-GeoMF}}$ or $\mathcal{E}_{\texttt{Cap-PMF}}$. The gradients can be obtained by a direct application of the chain rule. From (4.2), (4.3), (4.5) and (4.6), we get the gradients

$$\nabla_{\Delta} \ell(\Delta(c_j, \mathbb{E}[\text{usage}(j)])) = -\sigma(-\Delta(c_j, \mathbb{E}[\text{usage}(j)])) \tag{4.11}$$

$$\nabla_{\mathbf{u}_i} \Delta(c_j, \mathbb{E}[\text{usage}(j)]) = -p_i \mathbf{v}_j \sigma(\hat{r}_{ij}) \sigma(-\hat{r}_{ij}) \tag{4.12}$$

$$\nabla_{\mathbf{v}_j} \Delta(c_j, \mathbb{E}[\text{usage}(j)]) = -\sum_i p_i \mathbf{u}_i \sigma(\hat{r}_{ij}) \sigma(-\hat{r}_{ij}) \tag{4.13}$$

$$\nabla_{\mathbf{x}_i} \Delta(c_j, \mathbb{E}[\text{usage}(j)]) = -p_i \mathbf{y}_j \sigma(\hat{r}_{ij}) \sigma(-\hat{r}_{ij}) \ , \tag{4.14}$$

where for (4.12), (4.13), (4.14) we used the property $\nabla_x \sigma(x) = \sigma(x) \cdot \sigma(-x)$. Using (4.11)-(4.14), we obtain the gradient of the latent parameters as follows. The gradient of the objective, w.r.t $\mathbf{u}_i$ is

$$\begin{aligned}
\nabla_{\mathbf{u}_i} \mathcal{E}_{\texttt{capMF}} = &-(1-\alpha) \sum_{j \in L_i} 2(r_{ij} - \mathbf{u}_i^T \mathbf{v}_j) \mathbf{v}_j + 2\lambda \mathbf{u}_i \\
&+ \frac{\alpha}{N} \sum_{j=1}^{N} \sigma(-\Delta(c_j, \mathbb{E}[\text{usage}(j)])) \cdot p_i \mathbf{v}_j \sigma(\hat{r}_{ij}) \sigma(-\hat{r}_{ij}) \ .
\end{aligned}$$

Similarly, the gradient of the objective, w.r.t $\mathbf{v}_j$ is

$$\begin{aligned}
\nabla_{\mathbf{v}_j} \mathcal{E}_{\texttt{capMF}} = &-(1-\alpha) \sum_{i \in \text{Ra}(j)} 2(r_{ij} - \mathbf{u}_i^T \mathbf{v}_j) \mathbf{u}_i + 2\lambda \mathbf{v}_j \\
&+ \frac{\alpha}{N} \sigma(-\Delta(c_j, \mathbb{E}[\text{usage}(j)])) \cdot \sum_{i=1}^{M} p_i \mathbf{u}_i \sigma(\hat{r}_{ij}) \sigma(-\hat{r}_{ij}) \ .
\end{aligned}$$

Dataset	# Users	# Items	# Ratings
MovieLens 100K	943	1,682	100,000
MovieLens 1M	6,040	3,706	1,000,209
Foursquare	2,025	2,759	85,988
Gowalla	7,104	8,707	195,722

Table 4.1: Dataset Statistics.

For `cap-GeoMF`, the gradient of the objective, w.r.t $\mathbf{x}_i$ is

$$\begin{aligned}\nabla_{\mathbf{x}_i}\mathcal{E}_{\texttt{cap-GeoMF}} &= -(1-\alpha)\sum_{j\in L_i} 2(r_{ij} - \mathbf{u}_i^T\mathbf{v}_j - \mathbf{x}_i^T\mathbf{y}_j)\mathbf{y}_j + 2\lambda\mathbf{x}_i \\ &+ \frac{\alpha}{N}\sum_{j=1}^{N}\sigma(-\Delta(c_j, \mathbb{E}[\text{usage}(j)]))\cdot p_i\mathbf{y}_j\sigma(\hat{r}_{ij})\sigma(-\hat{r}_{ij}) \ .\end{aligned}$$

4.4 Experimental Results

We present empirical results to address the following questions:

1. What is the interplay of rating prediction and capacity loss for `Cap-PMF`, `Cap-GeoMF` as we vary the trade-off α? Similarly for pairwise ranking loss versus capacity loss for `Cap-BPR`, `Cap-GeoBPR`?

2. How do the proposed approaches `Cap-PMF`, `Cap-GeoMF`, `Cap-BPR`, `Cap-GeoBPR` compare with their state-of-the-art unconstrained counterparts?

3. How do *post-process* and *weighted objectives* solutions compare in terms of quality of recommendations at the top of the list?

4. How robust is our framework to different choices, i.e., surrogate loss for capacity loss, propensity choices, capacity choices, implicit vs. explicit data?

Data. For item recommendation, we considered two public real-world datasets containing users' ratings on movies: MovieLens 100K and MovieLens 1M; for POI recommendation, we used the Gowalla and Foursquare datasets. The users, items and ratings statistics of the data are found in Table 4.1, and preprocessing details can be found in Section 2.6.

Foursquare and Gowalla contain implicit check-in data (only values of 1 and 0 are present). A value of 1 denotes that the user has visited the POI while a 0 denotes that the user has not visited it; because the user potentially does not like the POI or does not know about it. MovieLens 100K and MovieLens 1M contain ratings in a multi-relevance scale of 1 to 5 stars. We experimented with two feedback setups for the MovieLens datasets: (i) two-scale explicit feedback (+1, -1), using 4 as the threshold of liking, and (ii) implicit feedback, marking every non-zero rating as 1, and keeping the rest as 0.

Evaluation Setup. We repeated all experiments five times by drawing new train/test splits in each round. We randomly selected 50% of the ratings for each user in the training set and moved the other 50% of each user's ratings to the test set. As the number of ratings per user vary, we chose this scheme to simulate the real recommendation setup: there exist users with a few ratings as well as users with many ratings.

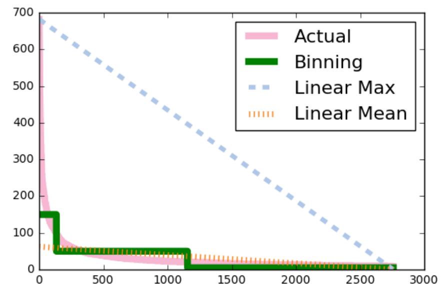
For the implicit feedback datasets, as only positive observations are available, we introduced negative observations (disliked items) in the training set as follows. For every user with N_i^{train} positive observations (+1), we sampled N_i^{train} items as negatives (-1) from the set of items marked as 0 both for train and test data. This is common practice to avoid skewed predictions from training on only positive observations, and to reduce the computational overhead of having all unrated items as negative observations.

Capacities and Propensities. The solution to the recommendation with capacity constraints problem relies on the availability of users' propensities (how likely they are to follow the recommendations); and of item/POI capacities (how many people are allowed to simultaneously use/occupy an item/place). As the information of capacities and propensities is not given in the considered data, and to the best of our knowledge to any of the publicly available recommendation datasets, we considered different simple ways of estimating them based on usage data.

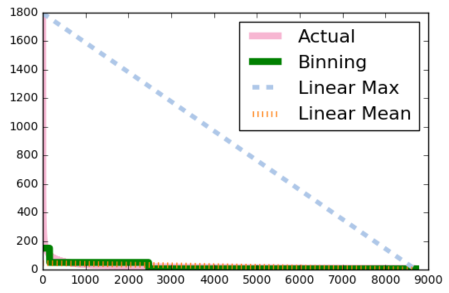
For capacities, we analyzed the three exhaustive cases: (i) capacities analogous to usage, i.e., such a setting is inspired by the supply-demand law: the more users ask for an item, the more copies of the item will be in the market, (ii) capacities inversely proportional to usage, i.e., capturing when items with low capacities are often in high demand, and (iii) irrespective of usage. In particular, we instantiated the above concepts with the following:

(C1) 'actual': $\forall j,\ c_j = \#$ users who have rated item j.[1]

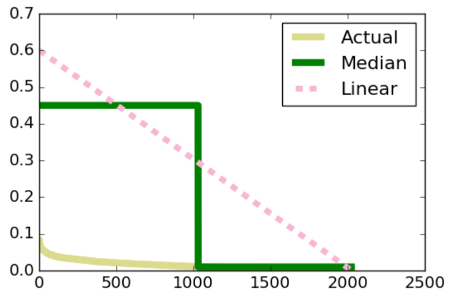
[1] An alternative would be # users who have liked the item in explicit data.



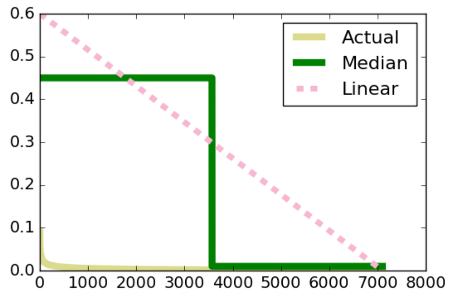
(a) Foursquare, capacities



(b) Gowalla, capacities



(c) Foursquare, propensities



(d) Gowalla, propensities

Figure 4.1: (a-b) Item capacities sorted in decreasing order, based on four different definitions. (c-d) User propensities, sorted in decreasing order, according to three definitions.

(C2) ‘binning’: Transform actual capacities to the bins: $[0, 20] \rightarrow 5$, $[21, 100] \rightarrow 50$, $[101, \text{max capacity}] \rightarrow 150$.

(C3) ‘uniform-k’: Set all item capacities to a value k, e.g. 10.

(C4) ‘linear max’: Spread the items’ capacities in [0, maximum actual capacity] using a linear function.

(C5) ‘linear mean’. Same as ‘linear max’, but spread in the range [0, 2*mean of actual capacities].

(C6) ‘reverse binning’: Map the actual capacities to the following bins : $[0, 20] \rightarrow 150$, $[21, 100] \rightarrow 50$, $[101, \text{max capacity}] \rightarrow 5$.

Note that (C1) -(C2) are analogous to usage, (C3) - (C5) are irrespective of usage and (C6) is inversely proportional to usage. Figures 4.1(a), (b) show all items' capacity scores sorted decreasingly for various capacity choices, for the POI datasets.

For user propensities, we considered the following cases: (i) user propensities are analogous to system usage by the user, and (ii) user propensities are irrespective of usage (i.e., capturing that propensity is an inherent user property). In particular, we considered:

(P1) 'actual':

$$p_i = \frac{\#\text{ observed ratings for user } i}{\text{Total }\#\text{ items}} = \frac{|L_i|}{N} \tag{4.15}$$

(An alternative could be: $p_i = \frac{\#\text{ liked items for user } i}{\#\text{ observed ratings for user } i} = \frac{|L_i^+|}{|L_i|}$.)

(P2) 'median': Set propensities $\geq$ the median of the actual propensities to 0.45, and propensities $<$ median to 0.01. This illustrates two distinct groups of users; those who tend to listen to the system's recommendations, and those who do not.

(P3) 'linear': Spread propensities in $[0, 0.6]$ using a linear function.

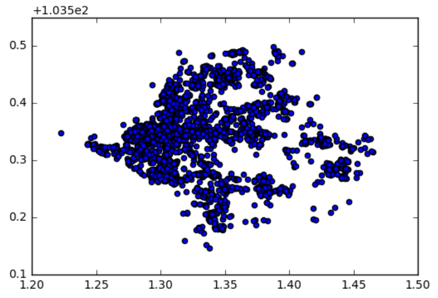
Note that (P1)-(P2) are analogous to user usage, while (P3) is irrespective of usage. Figures 4.1(c), (d) show users' propensity scores sorted decreasingly for the propensity choices in the POI data.

More intricate capacity and propensity estimation models can be considered as input to our framework as well.

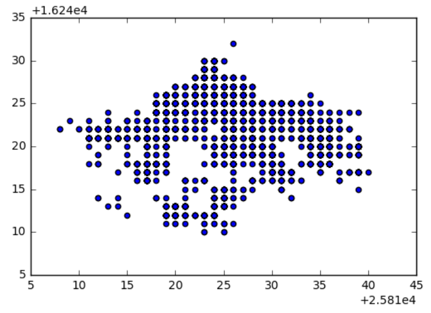
Location. The POI datasets Foursquare and Gowalla contain apart from the check-in observations, location information about where every POI lies in terms of geographical latitude and longitude coordinates. In Figures 4.2(a), (c) we show scatter plots of the latitudes (x axis) and longitudes (y axis) for the POIs of the non-subsampled datasets of Foursquare and Gowalla respectively.

Using the Mercator projection, and fixing the ground resolution to level of detail set to 15, we transformed the latitude/longitude coordinates of every POI to one of the 2^{15} tiles where the entire Earth can be divided into[2]. We show the tiles x-y coordinates of the various POIs for Foursquare in Figure 4.2(b) and for Gowalla in Figure 4.2(d). We set L, i.e., the dimension of the influence vector, to the number of unique tiles found.

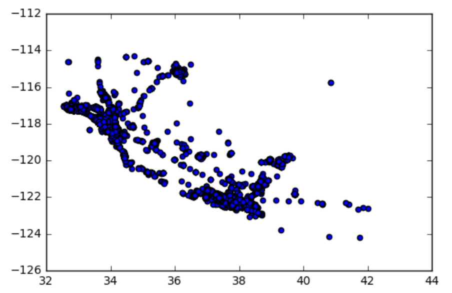
[2] https://msdn.microsoft.com/en-us/library/bb259689.aspx



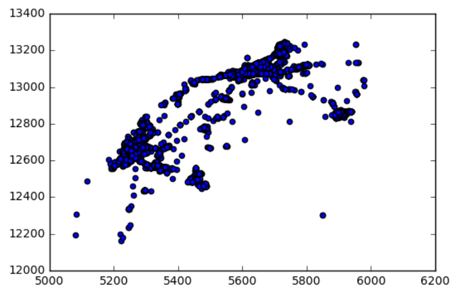
(a) Foursquare, lat./long. coordinates



(b) Foursquare, tile coordinates



(c) Gowalla, lat./long. coordinates



(d) Gowalla, tile coordinates

Figure 4.2: Location information scatter plots.

For Foursquare L is 290, whereas for Gowalla L is 4320. Considering the set of all L unique tiles and representing every POI as a pair of (tileX, tileY) coordinates, we pre-computed the influence matrix $Y \in \mathbf{R}^{L \times N}$ using Kernel Density Estimation [182], following the procedure briefly described in Section 4.2.3.

Parameter Setting. We stopped the training of the algorithm either when the improvement in the value of the optimization objective in the training set was smaller than 10^{-5} or after 3,000 iterations. Similarly to [103], we set the rank of the latent factors k to 10. We used for the learning rate of gradient descent the Adagrad rule [60], which for the latent factor of the user i, $\mathbf{u}_i$ at iteration t is: $\eta_t = 1/\sqrt{\sum_{\tau=0}^{t-1} \nabla^2_{\mathbf{u}_{i,\tau}}}$. We fixed the regularization parameter λ to 10^{-5}. We chose these parameters because we found they gave stable model performance. In practice, to further improve the performance, the parameters of rank and regularization can be tuned in a validation set.

Metrics. For the *weighted objectives* solutions of `CapMF`, `Cap-(Geo)BPR`, we report metrics relevant to the losses they optimize, to study the interplay of the two objectives. Particularly, we report test-set performance in terms of:

- For `Cap-PMF` and `Cap-GeoMF`, we report *Root Mean Square Error (RMSE)* which measures test set rating prediction accuracy (Section 2.4).
- For `Cap-BPR` and `Cap-GeoBPR` we report *0/1 Pairwise Loss* which measures the average number of incorrectly ordered pairs, i.e., -1 ranked above +1 (Section 2.4).
- *Capacity Loss*, which measures on average the extent to which the recommendations lead to violating the capacity constraints:

$$\text{Capacity Loss } = \frac{1}{N}\sum_{j=1}^{N} \ell\left(c_j - \sum_{i=1}^{M} p_i \sigma(\hat{r}_{ij})\right) , \tag{4.16}$$

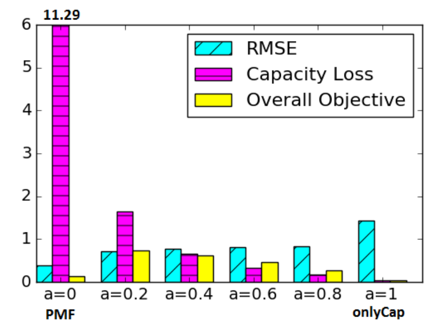
  with ℓ given by (4.6) or (4.7). For all three above metrics, values closer to 0 are better.
- *Overall Objective.* Given that our methods optimize the two objectives of rating prediction (`CapMF`)/ranking (`Cap-BPR`) loss and capacity loss, we also report: $(1-\alpha)\text{RMSE}^2 + \alpha\text{Capacity Loss}$ for `CapMF` (or $(1-\alpha)0/1$ Pairwise Loss $+$ αCapacity Loss for `Cap-BPR`), which captures the overall loss in the test set.

To compare methods overall, we use as a criterion the quality of recommendations at the top of the list, as measured by *AP@top* (Section 2.4).

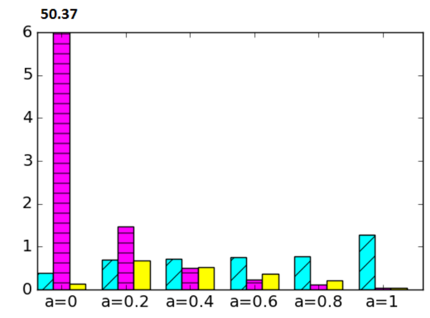
Methods Compared. We compare our methods with the baselines of: `PMF` [148], `BPR` [141], `GeoMF` [108], `GeoBPR`, and `onlyCap`, i.e., the baseline of setting α to 1. Also, we compare our proposed *weighted objectives* methods with the *post-process* methods.

4.4.1 Validation: Interplay between Objectives

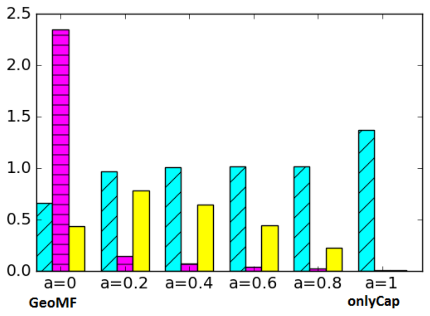
In the first set of experiments, our goal is to validate that indeed, using the trade-off parameter α, one can effectively balance recommendation performance and capacity loss. We expect that as α approaches 0, the algorithm will have better predictive performance but will be violating more the capacity constraints. In contrast, as α approaches 1, we expect the algorithm to respect more the capacity constraints at the cost of worse



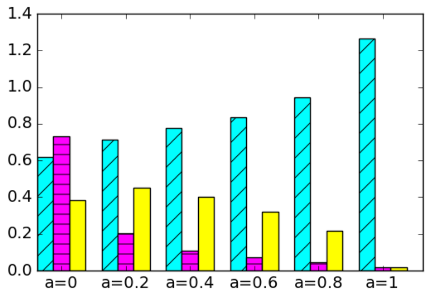
(a) MovieLens 100K



(b) MovieLens 1M



(c) Foursquare

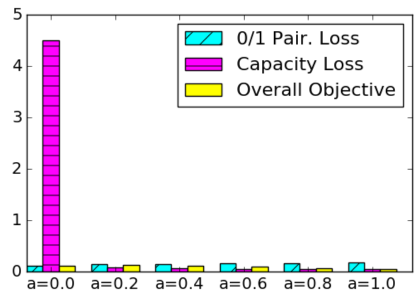


(d) Gowalla

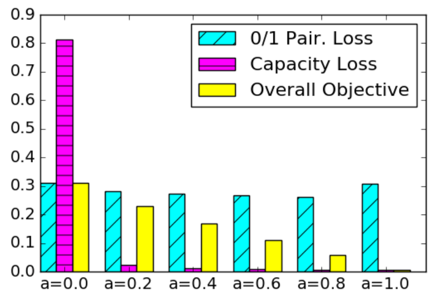
Figure 4.3: Effect of capacity trade-off parameter α in range $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$ on `CapMF`'s performance in test RMSE, Capacity Loss and Overall Objective. As expected, the higher the α, the higher the RMSE and the lower the Capacity Loss.

predictive ability. To illustrate this, we vary the trade-off parameter in $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$. We set the surrogate loss for the capacity objective to the logistic loss, and use the 'actual' capacity and propensity definitions.

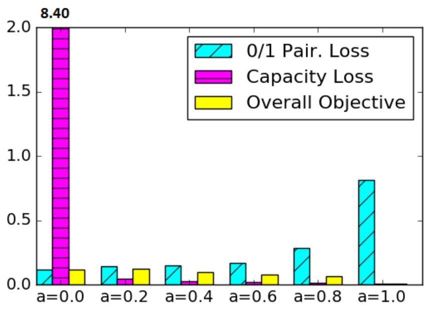
Rating Prediction vs. Capacity Loss: In Figure 4.3, we report the test set metrics of RMSE, Capacity Loss and Overall objective for all four datasets. The results show that indeed the more we increase the trade-off parameter, the smaller the capacity loss and the higher the RMSE. This validates that α can be used to specify to what extent the rating prediction accuracy is more/less important compared to the capacity loss for the considered application domain.



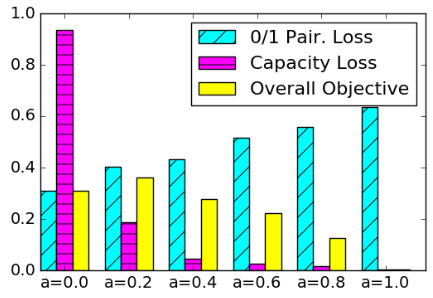
(a) MovieLens 100K



(b) Foursquare



(c) MovieLens 100K



(d) Foursquare

Figure 4.4: Effect of capacity trade-off parameter α in range $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$ on `CapBPR`'s performance in test 0/1 Pairwise Loss, Capacity Loss and Overall Objective. (a), (b): Using 'Actual' capacity definition. We observe that the 0/1 Pairwise Loss does not increase much as we increase α. (See why in Figure 4.5.) (c), (d): Using 'Reverse Binning' capacity definition. We observe the expected trade-off between ranking loss and capacity loss.

Ranking vs. Capacity Loss: Considering as first objective the pairwise ranking objective, in Figures 4.4(a), (b), we show the results of `Cap-BPR`/`Cap-GeoBPR` for MovieLens 100K and Foursquare, as we vary the trade-off parameter in the range of [0, 1]. Interestingly, we observe that while as expected, the capacity loss decreases with the increase of α, the 0/1 pairwise loss does not change much. Similar trends hold for the rest of the datasets (not shown). We found that the reason why this happens is that when the capacities are analogous to usage (here we used the 'actual' capacity definition), even if α is set to 1, i.e., not optimizing at all for ranking accuracy, the 0/1 pairwise training loss still decreases (Figure 4.5). This shows that when capacities are analogous

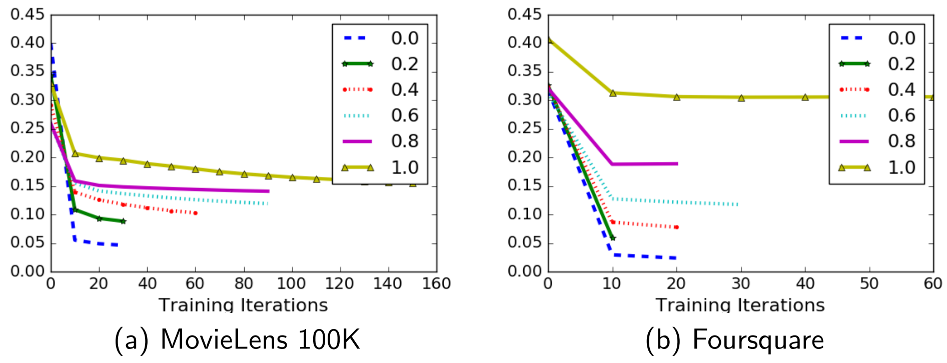


Figure 4.5: Training Pairwise 0/1 Loss of (a) `Cap-BPR`, (b) `Cap-GeoBPR` for 'actual' capacities, varying trade-off $\alpha \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$.

to usage, the capacity loss is related to the pairwise loss; namely, the capacity loss can help reconstruct the correct pairwise accuracy given only the item capacities and user propensities, with no other access to user-item ratings. Thus, the aggregate statistics of 'actual' user propensities and 'actual' item capacities act as sufficient statistics.

To see the expected trade-off between the objectives of capacity and ranking loss, we report in Figures 4.4(c), (d) `Cap-BPR`'s results for item capacities set inversely proportional to usage, using the 'reverse binning' capacity definition. Indeed, in this case, the fraction of incorrectly ordered pairs increases with the increase in α.

4.4.2 Comparison with Unconstrained Methods

For the next experiments we set α to 0.2, as we found it can lead to a reasonable trade-off among the objectives. In practice, α should be tuned in a validation set, to meet the desired trade-off of capacity loss vs. recommendation performance per application domain.

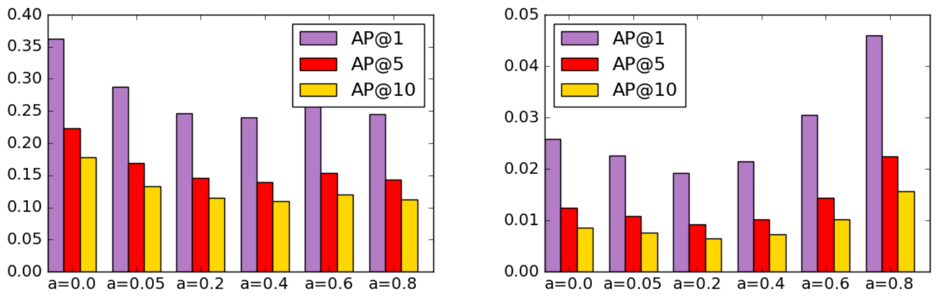
Comparison with `PMF`/`GeoMF` in RMSE/Capacity Loss: Based on Figure 4.3, considering the left end of the spectrum for the value of the trade-off parameter, i.e., $\alpha = 0$, we can compare our algorithm `CapMF` with the unconstrained methods `PMF`/`GeoMF`. We observe that: (i) For item recommendation, `Cap-PMF` significantly improves over `PMF`'s Capacity Loss performance of 11.29 to 1.65. This happens though at the cost of deteriorated performance in terms of RMSE from .38 to .71. Similar is the trend for MovieLens 1M. (ii) For POI recommendation, we observe that for Foursquare, `Cap-GeoMF` improves over `GeoMF`'s Capacity Loss performance of 2.35 to .15, at the cost of RMSE which increases from .66 to .97. Similar trends hold for Gowalla.

Comparison with BPR/Geo-BPR in Pairwise Loss/Capacity Loss: From Figures 4.4(a), (b), we can compare our method `Cap-BPR` with `BPR` and `GeoBPR` respectively. We see that for MovieLens 100K, `Cap-BPR` achieves Capacity Loss of 0.08 compared to 4.51 of `BPR`, while it results in 0/1 Pairwise Loss of .14 compared to .12 (higher values are worse). For Foursquare, `Cap-GeoBPR` achieves .02 Capacity Loss compared to .81 of `GeoBPR`, and also achieves a better 0/1 Pairwise Loss of .28 compared to .31. Similar trends hold for the other datasets.

Comparison with `onlyCap`: Focusing now on the other end of the spectrum of the trade-off parameter, i.e., $\alpha = 1$, we compare `CapMF` with `onlyCap`. We can see from Figure 4.3(a) that for MovieLens 100K, `onlyCap` improves Capacity Loss from 1.65 to .04, but results in RMSE from .71 to 1.43. Also, `onlyCap` results in a worse 0/1 pairwise loss of .17, compared to .14 of `Cap-BPR`, as seen from Figure 4.4(a).

Top-N Recommendations: Here, we evaluate the top recommendation quality using $AP@top$ as our metric, varying top in $\{1, 5, 10\}$.

From Figure 4.6 we see that for `CapMF`, and 'actual' capacities and propensities: for (a) MovieLens 1M, `Cap-PMF` has lower $AP@top$ compared to `PMF`, while for (b) Foursquare, `Cap-GeoMF` achieves higher $AP@top$ compared to `GeoMF`. We observe that



(a) MovieLens 1M (b) Foursquare

Figure 4.6: Average Precision (AP)@$\{1, 5, 10\}$ of `CapMF`.

as α increases, the top-N recommendation performance potentially improves, indicating that emphasizing more the capacity loss can further make the quality of the recommendations better. This aligns with our finding that the capacity loss can relate to the pairwise ranking loss, using the sufficient statistics of capacity and propensity. Also, as expected, while we increase the *top*, $AP@top$ decreases.

4.4.3 Comparison between Proposed Approaches

In this set of experiments, we compare our *weighted objective* methods (Section 4.3.2) with the *post-process* methods (Section 4.3.1). Recall that the *post-process* methods, by design, do not violate the capacity constraints. For the purposes of the novel setting of recommendation with capacity constraints, we propose a new metric:

$$\text{Weighted Average Precision @ top } (\mathit{WAP@top}) = \frac{\sum_{i=1}^{M} p_i AP_i@top}{\sum_{i=1}^{M} p_i}, \tag{4.17}$$

where AP_i is user i's Average Precision. $\mathit{WAP@top}$ measures recommendation quality in the *top* taking into account the users' propensities, by using propensities as weights in the weighted average of users' average precisions. We report in Table 4.2 the comparison of the *post-process* methods, using the best performing tuned α, with our proposed *weighted objectives* methods, and the respective unconstrained methods. For this experiment, we use Foursquare and explicit MovieLens 100K, and consider as underlying models `GeoMF`/`GeoBPR` and `PMF`/`BPR` respectively.

Based on Table 4.2, we make the following overall observations:

- For square loss, when capacities are proportional to usage ('actual', 'binning'), `Cap-PMF` outperforms `PostMF`, and even more interestingly outperforms `PMF` for *top* = 50. However, when the capacities are inversely proportional to usage, or are uniform, our proposed weighted objectives solution outperforms `PostMF`, but cannot outperform the unconstrained method. We can also see that for any choice of capacity, for geographically-aware methods, `Cap-GeoMF` outperforms both `Post-GeoMF` and `GeoMF`.
- For ranking loss, when capacities are proportional to usage, similar to square loss, `Cap-BPR` outperforms post-process, and can even outperform `BPR`. When capacities are uniform, `Cap-BPR` outperforms `Post-BPR`, and for item recommendation can even improve upon the unconstrained method. In contrast, for capacities inversely

		Square Loss						Pairwise Ranking Loss					
	Capacity Def.	WAP@10			WAP@50			WAP@10			WAP@50		
		`MF`	`PostMF`	`CapMF`	`MF`	`PostMF`	`CapMF`	`BPR`	`PostBPR`	`Cap-BPR`	`BPR`	`PostBPR`	`Cap-BPR`
ML 100K	Actual	0.152	**0.153***	0.138	0.086	0.087	**0.127***	0.115	0.117	**0.209***	0.069	0.07	**0.126***
	Binning	**0.152**	0.15*	0.136	**0.086**	0.077	**0.086***	0.115	0.112	**0.209***	0.069	0.064	**0.136***
	Reverse	**0.152**	0.045	0.072*	**0.086**	0.013	0.031*	**0.115**	0.04*	0.02	**0.069**	0.012*	0.01
	Uniform	**0.152**	0.055	0.135*	**0.086**	0.014	0.062*	0.115	0.055	**0.137***	0.069	0.013	**0.078***
Foursquare	Actual	0.016	0.016	**0.041***	0.011	0.011	**0.027***	0.056	0.049	**0.084***	0.033	0.023	**0.048***
	Binning	0.016	0.016	**0.023***	0.011	0.009	**0.014***	0.056	0.04	**0.084***	0.033	0.017	**0.047***
	Reverse	0.016	0.011	**0.039***	0.011	0.005	**0.015***	**0.056**	0.016*	0.013	**0.033**	0.005	0.007*
	Uniform	0.016	0.011	**0.047***	0.011	0.004	**0.024***	**0.056**	0.016	0.039*	**0.033**	0.005	0.023*

Table 4.2: Comparison of *weighted objectives* methods (denoted with `Cap-`) with *post-process* methods (denoted with `Post`), in terms of Weighted Average Precision@*top*, $top = \{10, 50\}$, for various capacity settings. Bold letters denote the best among the unconstrained, post-process, and `Cap-` methods. Asterisk (*) shows which capacity constrained method, i.e., `Post-` or `Cap-`, is better. Overall, the `Cap-` methods largely outperform the respective `Post-` methods. Interestingly, in some cases, `Cap-` methods outperform the unconstrained methods, showing the value of our method to even improve the recommendation quality.

proportional to usage, `Cap-BPR`, `Cap-GeoBPR` tends to be outperformed by the other methods.

Similar trends can be observed for non-weighted Average Precision, and for the other datasets, as well. Overall, these results highlight that our *weighted objectives* methods largely outperform the *post-process* solution. We can also draw the further conclusion that out of the alternative definitions considered for capacity, the definitions which are analogous to usage, namely 'actual' and 'binning', are the ones which lead to the best top-N recommendation results. In particular, under these settings the *weighted objectives* approach surpasses the *post-process* solution, and also is better than the unconstrained methods. This shows that these capacity settings can be realistic, and are worth incorporating in the optimization objective of the recommendation algorithms, to further improve top-N performance.

4.4.4 Sensitivity Analysis

Effect of Surrogate Loss for Capacity Term. Figure 4.7 compares the effect of different surrogate losses for the capacity objective on `CapMF`'s performance. For this experiment, we consider rating prediction as the first objective, we set α to 0.2, and use the 'actual' propensity and capacity definitions. We observe that for MovieLens 100K and MovieLens 1M (omitted), logistic and hinge loss result in similar performance,

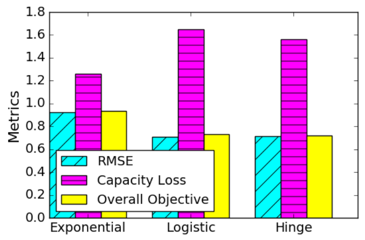


Figure 4.7: MovieLens 100K. Effect of surrogate loss for capacity term.

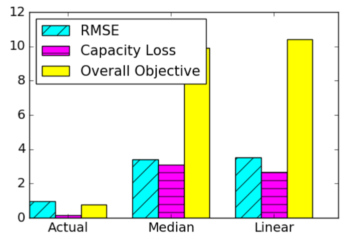


Figure 4.8: Foursquare. Effect of user propensities on `CapMF`.

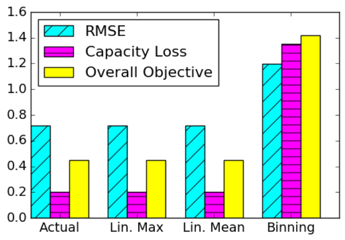
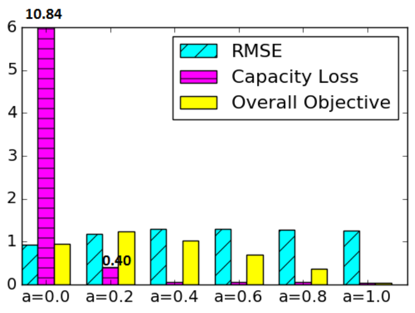


Figure 4.9: Gowalla. Effect of item capacities on `CapMF`.

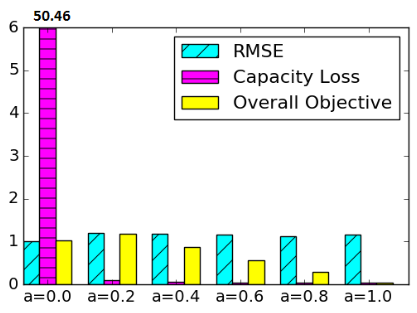
while exponential loss results in the highest RMSE and the smallest capacity loss. For Foursquare and Gowalla (omitted), hinge loss obtains the smallest capacity loss and RMSE, making it the best option for the POI datasets. This happens since the exponential loss assigns very high penalties for large negative differences. Also, exponential and logistic losses assign small penalties for small negative and also positive values, while hinge does not penalize when expected usage is within the capacity constraints.

Effect of User Propensities. In Figure 4.8 we study the effect of different user propensities as input to our algorithm (i.e., 'actual', 'median' and 'linear') on `CapMF`'s performance in the Foursquare dataset. For this experiment, we set α to 0.2, the surrogate loss to the logistic loss, the capacities to 'actual' and the first objective to rating prediction. We see that for 'median' and 'linear' choices of propensity, the values of both RMSE and Capacity Loss are higher compared to those obtained for the 'actual' propensity choice. This happens because the 'median' and 'linear' options generally result in higher values of user propensities (Figure 4.1(c)). This leads to higher expected usage values, making it more likely to have the capacity constraints violated.

Effect of Item Capacities. Here we study the effect of different item capacities, namely 'actual', 'linear mean', 'linear max' and 'binning', as input to `CapMF`. The setting is the same as before, except for the propensity definition which is set to 'actual'. Figure 4.9 compares `CapMF`'s performance for four choices of capacity for Gowalla—similar trends hold for the other datasets. We can see that the choice of 'binning' results in the highest RMSE and Capacity Loss, whereas the other choices result in almost identical performance. We explain this result as based on Figure 4.1(b) the 'binning' definition



(a) MovieLens 100K



(b) MovieLens 1M

Figure 4.10: `Cap-PMF` on explicit feedback data. The results shown compared to Figures 4.3(a), (b) indicate that the trends for implicit and explicit feedback data are similar.

typically results in the smallest items' capacities, which means that it is more likely that the recommendations will violate the capacity constraints. Also, when setting all items' capacities uniformly to e.g. the mean actual capacity (not shown), the capacity constraints are directly satisfied.

Implicit vs. Explicit Feedback: Finally, we report in Figure 4.10 the results of `Cap-PMF` in terms of the competing objectives for the original explicit MovieLens datasets. This allows us to explore how sensitive our framework is to implicit versus explicit data. Thus, we can compare Figure 4.10(a) with Figure 4.3(a) for MovieLens 100K, and Figure 4.10(b) with Figure 4.3(b) for MovieLens 1M. We can see that although the particular values of the objectives are different, the trends are similar. Similar trends were also found for `Cap-BPR`.

4.5 Related Work

Although we could not trace literature handling the problem of *recommendation with capacity constraints*, there are works using the notion of constrained resources or budget [80]. Here, we present how each of these settings presents unique challenges, and differs from our problem.

Email Volume. One related constraint which has been recently explored by Gupta et al., and posed as a Multiple Objectives Optimization (MOO) problem [76], is deciding which emails to send so to minimize the total number of emails, while the number of downstream sessions is maximized, and the number of spam reporting and unsubscribe actions is minimized. While email volume can be seen as a capacity constraint, our work is rather different: In our setting, the end goal is to provide *personalized recommendations* which respect the capacity constraints, whereas in [76] a non-personalized email allocation scheme is proposed.

Budget. The problems of sponsored search [14], advertisement display [101, 116], auction [180, 102] etc., are naturally associated with the notion of a budget. For instance, for ad displays, the algorithm decides which ads to show so that the number of clicks or revenue is maximized, while the advertisers stay in their specified budget. Typically, such problems have been formulated using online matching [116], or bandits [180, 102]. Our setting departs from the above problem in a number of ways: (i) Items have a budget, instead of users having a budget. (ii) We collaboratively allocate items to users in a unified way; in contrast, usually, ad serving systems have separate entities for ad bidding and for serving personalized ads to users.

Course Requirements. In [131] the authors propose novel methods for set recommendations of courses, so that the courses recommended to a student satisfy the specified degree requirements. In our work, instead of focusing on constraints within the list of items, we focus on satisfying the capacity constraint coupling recommendation lists across users as a whole.

Quantity constraints. Zhang et al. proposed to boost the Pareto efficiency of web allocations [183]. While they also provide personalized allocations modeling the web economy as a whole, their focus is rather different from ours; they maximize each user's individual surplus, while we focus on maximizing recommendation performance, as typically captured in recommender systems. Also, they require the product prices as input, which is not applicable in our case. Our work is the first to introduce the novel aspect of the tendency of users to follow the system's recommendations, giving different weights to the various users when estimating item usage, and to provide personalized allocations that also account for geographical influence.

4.6 Summary

We have presented a novel approach for providing recommendations that satisfy capacity constraints. We have demonstrated how this generic approach can be applied to three state-of-the-art latent factor models, `PMF` [148], `GeoMF` [108], and `BPR` [141], so that the items' expected usage respects the corresponding capacities. Our experimentation has given light into how our methods perform under different settings of item capacities and user propensites. We have shown that our framework effectively provides recommendations which respect the capacity constraints, without deteriorating the recommendation performance by a lot; interestingly, in some cases our methods can improve the top-N recommendation quality compared to the respective unconstrained methods.

Part II

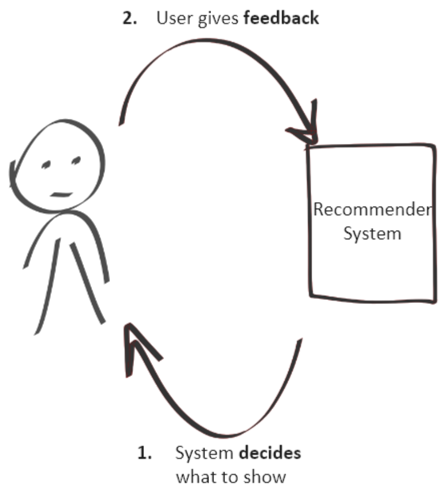
Data: Learning To Interact with Users

Based on work previously published in
KDD 2016 [51] and SDM 2018 [48].

Part II: Overview

This part addresses the question: where does the data with which we train the recommendation model come from? More specifically, how can we answer questions of the form: what will happen if we show an item j to user i?

To answer this question, the *offline model* view used in the majority of works on recommendation, including the previous part of this dissertation, is insufficient; as it can only capture associational relationships among variables based on given data. This static view (i) suffers from the cost of periodically retraining the model to incorporating new observations, (ii) fails to capture preference drifts, and (iii) has the inherent limitation that what is built to optimize offline performance does not necessarily translate to equally good online user behavior.


2. User gives **feedback**
Recommender System
1. System **decides** what to show

Interactive learning: User in the loop in a recommendation system.

Instead, in the following part we will adopt a modern approach of viewing recommendation as an interactive learning problem [104, 165], which can allow the making of interventions—changing what we see instead of just observing what is there—and which models the presence of limited information (*bandit fedback*): when showing item j to user i, we do not know what would have happened had we shown something else. However, because of the large dimensionality of the user and item space, classic off-the-shelf *multi-armed bandit* techniques are not directly applicable. Thus, in both chapters of this part, we will couple bandit techniques with collaboratively learning on-the-fly the structure among users and items.

We postulate that modeling recommendation in an online learning framework while balancing the explore-exploit trade-off present in any system learning to interact, can lead to better understanding of the user's preferences, and thus improved recommendation quality. Specifically, we offer two contributions:

- **Learning to converse with new users.** By asking a cold-start user a couple of questions, either of absolute-type (Do you like restaurant j?) or of relative-type (Do you prefer restaurant j over restaurant h?), not only can the system learn more about the user's preferences and about what question can more quickly allow the system to give a good recommendation, but also we make a step towards modeling how a real person would make a recommendation to someone they do not know.
- **Learning to collaboratively interact with users.** We find that by coupling Thompson Sampling (i.e., a bandit technique that uses the underlying model uncertainty to balance the explore-exploit trade-off) with a latent-factor based collaborative filtering model, we can achieve improved online system performance, when the number of users and the size of the pool of items is large.

Chapter 5

Recommendation as a Conversational Procedure

Can we create human-like recommenders
that represent how people give recommendations?

People often ask others for restaurant recommendations as a way to discover new dining experiences. This makes restaurant recommendation an exciting scenario for recommender systems and has led to substantial research in this area. However, most such systems behave very differently from a human when asked for a recommendation. The goal of the work presented in this chapter is to begin to reduce this gap.

In particular, humans can quickly establish preferences when asked to make a recommendation for someone they do not know. We address this cold-start recommendation problem in an online learning setting. We develop a preference elicitation framework to identify which questions to ask a new user to quickly learn their preferences. Taking advantage of latent structure in the recommendation space using a probabilistic latent factor model, our experiments with both synthetic and real world data compare different types of feedback and question selection strategies. We find that our framework can make very effective use of online user feedback, improving personalized recommendations over a static model by 25% after asking only 2 questions. Our results demonstrate dramatic benefits of starting from offline embeddings, and highlight the benefit of bandit-based explore-exploit strategies in this setting.

5.1 Introduction

Recommendation is an everyday process that frequently touches people's lives. Hence, it has seen tremendous research interest (such as [57, 94]). Most work in recommendation falls into two broad classes: *Collaborative Filtering* starts with a set of user/item affinity scores and assumes that two users who agree about one item are more likely to agree about another item; *Content-Based Filtering* builds user or item profiles based on external feature information. We note that neither model represents how real people make recommendations, particularly in a cold-start setting where the person making a recommendation does not know a lot about the person asking for one.

Consider what would happen if a conference attendee in your home town, whom you have never met before, asked for a recommendation on where to eat dinner today. Most likely, you would start with one or two clarifying questions, perhaps whether the person likes seafood, or whether they have a car. These questions would depend on the context; for instance if there are great restaurants around the corner, then whether they have a car would be irrelevant.

We argue that such an interaction can be represented using online learning, where two types of learning are occurring. First, the person making the recommendation is learning about the preferences of the person asking. However, the attributes learned will be *contextual*, based on the likely follow-on answers (such as the car question earlier). Second, the person making the recommendation is learning about which questions allow them to quickly reach a good recommendation in the current context. In this chapter, we present a recommendation system that exhibits these two types of learning. Further, the learning is online: It immediately impacts future recommendations for this user, rather than requiring a batch reprocessing of information learned.

We present a bandit-based approach for online recommendation, applied to restaurant recommendation so as to ground it in a specific application. Our approach builds on top of prior work, such as [184], but learns to adapt the recommendation space (user-item embedding) to its users throughout their interactions. We use generalized Thompson Sampling to systematically sample questions to ask the user and to incorporate observed feedback. We further propose and compare a range of alternative question selection strategies to identify characteristics of approaches that most effectively learn users' preferences. We use a matrix factorization approach [118, 160] to learn and adapt the embedding.

Our main contributions are four-fold:

1. We propose a novel view of human-like recommenders that converse with new users to learn their preferences.

2. We successfully demonstrate a fully online learning approach for recommendation—both using absolute and relative feedback.

3. We propose a systematic approach to incorporating offline data to initialize online learning recommenders, and demonstrate performance improvements, even using weakly labeled offline data.

4. We propose a set of item selection strategies for deciding what question to ask to a cold-start user to most quickly infer preferences, and demonstrate benefits of bandit-based strategies.

Our extensive experiments on both synthetic and real data evaluate each step of our approach.

Importantly, we note that this work is applicable to a wide variety of recommendation scenarios. Here, we focus on one such application, restaurant recommendation, where we study search logs to understand the space of real user needs; we use the insights to collect preference data in a user study; we use online behavioral data to initialize the recommendation system; we use both synthetic and user study data to evaluate our approaches.

We now present an analysis of real-live restaurant search (Section 5.2). Next, we describe our model (Section 5.3) and empirical setup (Section 5.4), and validate the model on synthetic data (Section 5.5). We evaluate on real data to further empirically analyze our learning framework (Section 5.6). Finally, we review related work (Section 5.7), and summarize our work (Section 5.8).

5.2 Understanding Real Users

A recommendation system that aims to satisfy real people should reflect how real people look for recommendations. Therefore, we start by analyzing restaurant-related search behavior in a commercial Web search engine. Our goals are two-fold. First, to gain insight into our target domain, namely understand the criteria people use to choose restaurants[1]. Second, to identify questions we must ask users to construct ground truth

[1] Keeping in mind that these criteria to some degree reflect users' perception of the search engine and its capabilities.

data for evaluating our system.

Given a large sample of all queries issued between 07/2014 and 07/2015, we filtered for those that contain *restaurant* or *dining*, and terms such as *for*, *near(by)*, *next (to)*, *close (to)*, *with* and *in*. Let $\mathcal{Q}$ be the most frequent 10,000 such queries.

5.2.1 Query Annotation with Entities

We tagged the top several hundred queries from $\mathcal{Q}$ as containing a location, name, cuisine, food type, or terms such as *best* and *menu*. The dictionary of tags was not pre-specified to avoid biasing our annotations due to prior beliefs about categories people *should* be looking for. Rather, our methodology used content analysis [144] to develop a coding that captures the dominant behaviors involved.

We found that 39% of queries specify a location, 19% a restaurant name, 9% cuisine constraints, 7% have the term *best* or other superlative. More rarely, queries specified food ethnicity (2%), an adjective describing the restaurant (2%), dish name (2%), decoration, etc. The most common term co-occurences are location and name in the same query (29%), cuisine and location (10%), and *best* and location (8%).

5.2.2 Understanding Common Phrases

The above statistics show that location is an important factor in restaurant search. Hence we zoom in to discover the specific ways in which people constrain locations. Similarly, the context under which users search for a restaurant is a second common constraint. We analyzed the specific terms appearing after a variety of prepositions, and constructed a dictionary of the most frequent contextual constraints.

A sample of these is shown in Table 5.1. For example, the top places people search for restaurants *in* are nyc, chicago, las vegas. Using the preposition 'near' to indicate location, the majority of terms shows users want a restaurant *near me*. Queries can also be very specific, e.g. searching for a restaurant near points of interest (*times square*), airports (*miami airport, lax*), postal codes or entire addresses. The contexts under which users search for restaurants vary from an occasion (*wedding reception*), to dietary restrictions, to type of meal or a group of people. People also search for restaurants *with live music, piano* etc.

IN	**NEAR**	**FOR**	**WITH**
nyc	me	wedding receptions	party room
chicago	by	large group	small private room
las vegas	times square	rich	piano
houston	here	dessert	live music
miami	lax	your 21st birthday	someone
los angeles	miami airport	steak at lunch time	a view
atlanta	airport	gluten free food	play area
brooklyn	fenway park	valentine day in dallas	banquet room

Table 5.1: Contextual terms in restaurant related queries.

Cuisine	Freq.
mexican	475
chinese	407
italian	381
thai	174
indian	144
japanese	125

Adj.	Freq.
good	27
famous	13
nice	12
romantic	11
upscale	6
small	6

Food	Freq.
seafood	139
sushi	40
steak	32
bbq	29
fish	24
tapas	19

Table 5.2: Restaurant Feature Dictionaries (note: frequency counts have been rescaled).

5.2.3 Restaurant Feature Dictionaries

As user queries can be very specific (e.g., combining constraints, ‘adjective & dish & great & location’), we now study the terms that people use to describe restaurants. We annotated the top 1,013 phrases appearing before the word ‘restaurant’ or ‘dining’ in 5,000 queries from $\mathcal{Q}$ with a number of traits. The traits identified are related to food (cuisine, meal, dietary restrictions, quality e.g. *organic, healthy*, buffet, menu), rating (e.g michelin star), atmosphere (view, *fancy* or *romantic*), time/location (opening hours, decade theme, time of the day) etc.

For every trait, we collected the most common terms. Table 5.2 shows a few common examples, giving us a glimpse into the most searched cuisines, food types, and adjectives. Though not shown, top amenities searched are *garden, jazz, patio* and top restaurant types are *bar, fast food.*

5.2.4 Outlook

Besides motivating our application scenario, this search log analysis forms the basis of the user study in Section 5.6. Given this understanding of *what to ask* people who look for a restaurant, we also focus on *how to ask it*. In this work, among the various combinations of the feedback type (explicit/implicit, absolute/relative/list) on the available content (explicit features/items/latent features), we elicit users' preferences using *explicit feedback* from *absolute* and *relative* questions about *explicit items* (e.g. restaurants).

5.3 Model

We next present our approach for determining 'what to ask' and 'how to ask' as the key pieces of a conversational recommender, starting with a high level picture of the entire algorithm (Section 5.3.1). Then, we describe the pieces in detail as we require (i) a model exploiting implicit structure among items and users to efficiently propagate feedback (Section 5.3.2); (ii) an explore-exploit approach to probe the space of items, to allow continuous learning, (Section 5.3.3) and (iii) a feedback elicitation mechanism for selecting absolute (Section 5.3.3) and relative questions (Section 5.3.4).

5.3.1 Overview

Our recommendation pipeline can be summarized as:

1. Pick a model (Absolute/Pairwise) and preference elicitation mechanism: `Abs` / `Abs Pos` / `Abs Pos & Neg` / `Pairwise`.
2. Initialize model parameters using offline data.
3. A new user arrives. Now iterate for a few questions:
   (a) Mechanism selects a question to ask
   (b) User answers the question
   (c) All model parameters are updated
   (d) Remove the question from the allowed questions
4. System presents the final recommended list

The inner loop represents the 'human in the loop' present in all interactive systems, i.e., we make an intervention which affects the user and thus the future system decisions.

5.3.2 Latent Factor Recommendation

Consider how people make recommendations when a friend asks them to suggest a restaurant. They strategically ask each question with the goal of eliminating or confirming strong candidates for dining out. Similarly, when designing a conversational recommender that asks questions about explicit items, a principled way of selecting items is desired.

The implicit structure of the item space that allows us to learn quickly is motivated by *collaborative filtering*. Items that have been co-rated similarly (liked/disliked) by users will lie close in a low dimensional embedding. For our model we use a simplified version of the Matchbox Recommender model [160]—equivalent to `PMF` [118]. This is a generative model, in that it specifies a probabilistic procedure by which the observed likes/dislikes of users on items are generated on the basis of latent variables. The model variables are learned so that the model can explain the observed training data.

We use the convention that i denotes the index over M users, forming the set $\mathcal{U}$, and j denotes the index over N items, forming the set $\mathcal{I}$. Every user $i \in \mathcal{U}$ is modeled by a user bias variable $\alpha_i \in \mathbb{R}$, accounting for users who tend to like/dislike most of the items, and a d-dimensional trait vector $\mathbf{u}_i \in \mathbb{R}^d$. The trait vector represents the latent embedding of user i in a d-dimensional space, where $d \ll M, N$. Every item $j \in \mathcal{I}$ is modeled with latent variables $\beta_j \in \mathbb{R}$ (the item bias that accounts for item popularity) and a trait vector $\mathbf{v}_j \in \mathbb{R}^d$ that represents the latent embedding of item j in the *same* d-dimensional space. In the rest of the chapter, we use the notions of trait vectors and latent factors interchangeably.

Given that both users and items trait vectors lie in the same latent space, the similarity between a user i and an item j can be measured with the inner product of their corresponding trait vectors $\mathbf{u}_i^T \mathbf{v}_j$. We now present two models for estimating the latent variables, depending on the type of observations we obtain from users, i.e., absolute or pairwise.

Absolute Model. First, let us assume that we have observed tuples of the form (user i, item j, 1/0).[2] The model estimates the *affinity* of user i to item j based on the

[2] We use the convention that 0 denotes dislike and 1 like.

biases and traits. The generative procedure is:

1. User i has traits $\mathbf{u}_i \sim \mathcal{N}(\mathbf{0}, \sigma_1^2\mathbf{I})$, bias $\alpha_i \sim \mathcal{N}(0, \sigma_2^2)$.

2. Item j has traits $\mathbf{v}_j \sim \mathcal{N}(\mathbf{0}, \sigma_1^2\mathbf{I})$, bias $\beta_j \sim \mathcal{N}(0, \sigma_2^2)$.

3. (a) The (unobserved) affinity is

$$y_{ij} = \alpha_i + \beta_j + \mathbf{u}_i^T \mathbf{v}_j. \tag{5.1}$$

   Observations are modeled as the noisy estimate $\hat{y}_{ij} \sim \mathcal{N}(y_{ij}, \epsilon_{ij})$, where ϵ_{ij} models the affinity variance, accounting for noise in user preferences. This yields an observation of whether the user likes an item ($\hat{r}_{ij}$):

$$\hat{r}_{ij} = \mathbf{1}[\hat{y}_{ij} > 0]. \tag{5.2}$$

The hyper-parameters σ_1, σ_2 model the variance in traits and biases. The model variables are learned by maximizing the log-posterior over the item and user variables with fixed hyper-parameters, given the training observations.

Pairwise Model. People are often better at giving relative preferences over items instead of absolute judgments. Such preferences yield tuples of the form (user i, item j, item h) when user i prefers item j over item h (both j and h are indices over $\mathcal{I}$.) We can adapt the generative model to such ground truth data by modifying the third step of the *Absolute* model to yield a pairwise model:

3. (b) For each observation (i, j, h) compute

$$\text{noisy difference: } \hat{y}_{ijh} = \hat{y}_{ij} - \hat{y}_{ih} \tag{5.3}$$

   where the noisy item affinities are defined as before. Using $\hat{y}_{ijh}$, we estimate if user i prefers j over h:

$$j \succ_i h : \hat{r}_{ijh} = \mathbf{1}[\hat{y}_{ijh} > 0] \tag{5.4}$$

5.3.3 Continuous Learning Recommender

To generate recommendations, most latent factor-based recommender systems use an offline trained model based on past interactions, such as the one described above, that

Algorithm 2 Preference Elicitation Algorithm

```
Require: $\forall i \in \mathcal{U} : \mathbf{u}_i, \alpha_i$, $\forall j \in \mathcal{I} : \mathbf{v}_j, \beta_j$ (offline embedding)
1: A new user $i$ arrives. Initialize prior based on Eq. 5.5.
2: $\forall j \in \mathcal{I}$, infer noisy affinity $y_{ij}$ (Eq. 5.1)
3: while fewer than allowed questions have been asked: do
4:     Pick item(s) for absolute (relative) question (Sec. 5.3.3/ 5.3.4).
5:     Incorporate feedback according to (i) Abs (Sec. 5.3.3), (ii) Abs Pos, (iii) Abs Pos
       & Neg, or (iv) Pairwise. (Sec. 5.3.4)
6:     Update $\mathbf{u}_i$, $\alpha_i$, $\mathbf{v}$, $\beta$ (Section 5.3.3) and infer the noisy affinity distribution $\mathbf{y}_i$ (Eq.
       5.1) or noisy difference (Eq. 5.3).
7: end while
```

they periodically update to incorporate new data. The parameters for new users are typically initialized based on a combination of explicit attributes and past ratings (e.g [5]). The items with the highest predicted noisy affinity mean comprise the recommendations presented to the user.

In contrast, recent work has moved towards continuously learning recommenders [30, 104, 184]. This optimizes for *online* performance using explore-exploit (EE) strategies. We present our fully online updated recommendation approach with the embedded preference elicitation in Algorithm 2. Following, we detail the components of this algorithm for the case of asking absolute questions (`Abs`). In Section 5.3.4 we show how we extend this framework for relative questions.

Initialization from Offline Data

We propose to initialize online learning models using an initial embedding, that is learned offline. We hypothesize that such an initial embedding will allow the system to learn new user's preferences more quickly. Effective learning from few questions is crucial for conversational recommenders.

We start by learning the *offline embedding* of items from logged observations. Then, we initialize the prior of every item $j \in \mathcal{I}$ by setting the trait $\mathbf{v}_j$ and bias β_j from the corresponding offline posterior—assuming that all items in the online phase appeared in the offline data.

For the initialization of the user parameters, we focus on the case when the user is new to the system. Without additional information, we can assume that the new user is similar to the average offline user. We implement this by using as trait and bias the mean value over all offline users:

$$\mathbf{u}^{cold} \sim \mathbb{E}_{i=1,\ldots,M}[\mathbf{u}_i] \qquad \alpha^{cold} \sim \mathbb{E}_{i=1,\ldots,M}[\alpha_i] \tag{5.5}$$

Question Selection Strategies

When a new user initiates interaction with a continuous recommender, the system asks a few questions to learn about the user's preferences. During this phase, it is important to select questions that lead to learning effectively (i) the user's preferences and (ii) the questions' quality, so that the number of questions asked can be minimized and interactions remain enjoyable. This task can be modeled as an item selection task. Here, we propose approaches that capture characteristics of *active learning* and *bandit learning*. Active learning approaches capture the intuition that learning is fastest when the system queries for labels that provide a high amount of new information [145]. Bandit learning approaches balance the need to learn new information with a focus on what has already been learned [17, 38]. In the context of conversational recommenders, such a balance may help focus questions on the most relevant part of the latent space, while still considering that highly preferred items may lie in as of yet unexplored areas of the space.

The model's confidence in its belief over the user's preferences on items $j \in \mathcal{I}$ at a given time is captured by the current variances of the posterior of the noisy affinities y_j^{cold}. As we ask about an item j^* and observe the user's feedback, the variance of the inferred noisy affinity of this item *and of the nearby items* in the learned embedding is reduced. Also, the means of these items' inferred noisy affinities change. It is this property that allows us to search the space of items quickly. Hence, while in the classic multi-armed bandit scenario the bandit algorithms converge only after all arms are sufficiently explored, in our setting the collaborative structure allows for faster convergence.[3]

We compare a number of approaches for question selection that reflect the considerations discussed above, and several baselines. All approaches are listed in Table 5.3. Each selects an item j^* to obtain user feedback on. While the first few are self-explanatory baselines, we discuss three in more detail. (1) MaxT approximates maximizing information gain, in the vein of active learning: As user-item ratings are observed, the PMF

[3] A formal regret analysis lies beyond the scope of this work.

Greedy: $j^* = \arg\max_j y_{ij}$ A trivial *exploit*-only strategy: Select the item with highest estimated affinity mean.
Random: $j^* = \text{random}(1,N)$ A trivial *explore*-only strategy.
Maximum Variance (MV): $j^* = \arg\max_j \epsilon_{ij}$ A *explore*-only strategy, variance reduction strategy: Select the item with the highest noisy affinity variance.
Maximum Item Trait (MaxT): $j^* = \arg\max_j \|\mathbf{v}_j\|_2$ Select the item whose trait vector $\mathbf{v}_j$ contains the most information, namely has highest L2 norm $\|\mathbf{v}_j\|_2 = \sqrt{v_{j1}^2 + v_{j2}^2 + \ldots + v_{jd}^2}$.
Minimum Item Trait (MinT): $j^* = \arg\min_j \|\mathbf{v}_j\|_2$ Select the item with trait vector with least information.
Upper Confidence (UCB): $j^* = \arg\max_j y_{ij} + \epsilon_{ij}$ Based on UCB1 [17]: Pick the item with the highest upper confidence bound, namely mean plus variance (95% CI)
Thompson Sampling (TS) [38]: $j^* = \arg\max_j \hat{y}_{ij}$ For each item, sample the noisy affinity from the posterior. Select item with the maximum sampled value.

Table 5.3: Question selection strategies evaluated.

model adds information to the corresponding user and item trait vectors. In the opposite extreme case, if all item trait elements are close to 0, the corresponding item carries no information. As MinT does the opposite from MaxT, it is hypothesized to have low performance and is employed to establish a lower bound on question selection performance. (2) UCB [17] is a popular bandit algorithm that selects the items with the highest confidence bound to avoid missing preferences for promising items. (3) Thompson Sampling (TS) is a bandit algorithm that balances exploration and exploitation by selecting items using a sampling strategy [38]. It samples from its current posterior belief over noisy affinities, and then acts optimally according to this sampled belief. TS focuses on items with high mean affinity, but is also likely to select items with high variance.

Online Updating

After posing a question to the user, the observed response needs to be incorporated into to the recommender to allow for continued learning. As shown in [128], the questions can be incorporated into the model by setting the probability of the question to 1 and incorporating the user's response following standard probability theory.

The user's response thus becomes a new observation that the system uses to update the posterior distributions of *all* latent model parameters related to the incoming user i and the item j asked about, i.e., $\alpha_i, \beta_j, \mathbf{u}_i, \mathbf{v}_j$ (but affecting only user i's interaction session). To select the next question for user i, we use the updated posteriors as the new priors to infer the user's noisy affinity distribution $\hat{y}_{ij}$ for all items $j \in \mathcal{I}$, denoted by $\hat{\mathbf{y}}_i$. As the system keeps asking questions to user i and incorporates his/her feedback, the beliefs about the user, and the item in the question, change. This allows the model to move towards the true underlying affinity distribution. All online updates were implemented using expectation propagation [160] in Infer.NET [117].

Absolute Model, Absolute Questions (`Abs`): So far we have presented the entire framework for the case where the system poses absolute questions. Before turning to relative feedback, we describe the high-level approach for asking absolute questions (`Abs`). Using TS for illustration purposes, `Abs` asks user i about the item with the largest sampled noisy affinity as inferred by the *Absolute* model:

$$j^* = \arg\max_{j \in \mathcal{I}} \hat{y}_{ij} \tag{5.6}$$

Based on whether the user (dis)liked item j^*, a new observation (i, j^*,1/0) is incorporated into the Absolute model.

5.3.4 Extension to Relative Preferences

An alternative is for the system to ask for a preference about a pair of items, i.e., does the user prefer item A (j^*) or item B (h^*)? Therefore, we present here the extension of our framework to the case of asking *relative questions.*

We consider three separate formulations (referred to as `Abs Pos`, `Abs Pos & Neg` and `Pairwise`) for selecting relative questions and incorporating feedback, to identify the best way of asking relative questions. Importantly, in every such formulation there are two choices: (i) the underlying model (*Absolute* vs. *Pairwise*) and (ii) how the user's response to the question is incorporated back into the model. For the first choice, `Abs Pos` and `Abs Pos & Neg` use the Absolute model, while `Pairwise` uses the Pairwise model. The second choice is applicable only for `Abs Pos` and `Abs Pos & Neg`, as they represent two ways of incorporating relative feedback into the absolute model.

Absolute Model, Relative Questions.

First, we present `Abs Pos` and `Abs Pos & Neg`. Both use the same mechanism to generate the question "A vs. B" for user i:

1. Select item A as in `Abs` (Equation 5.6).
2. Virtual observation: Assume user i did not like A.
3. Virtual update: Incorporate the tuple (i, A, 0) into the *Absolute* model, infer the posteriors for all model parameters and set them as the virtual new prior.
4. Select item B, again according to `Abs`, but this time using the virtual prior as prior.

The insight behind this mechanism of constructing the relative question is that the two items the user is asked to give a relative preference on should be relatively far apart in the latent embedding, so that (i) the system can learn users' preferences effectively and (ii) the user is not forced to choose among very similar items. This diversity enforcing mechanism is inspired by the approach in [39].

The two methods introduced here differ only in the way the feedback is incorporated into the Absolute model. `Abs Pos` incorporates only positive information while `Abs Pos & Neg` incorporates both positive and negative information. For example, assume that the user preferred item B to A. Then, `Abs Pos` incorporates only the observation (i, B, 1), interpreting the relative preference on the preferred item B as a like for B. In contrast, `Abs Pos & Neg` incorporates two observations: (i, B, 1) for the preferred item and (i, A, 0) for the less preferred item. This can be seen as a variant of the "sparring" approach to dueling bandits [12], which samples item pairs and updates the model for both items as if an absolute reward were observed.

Pairwise Model, Relative Questions.

The third method for selecting relative questions, `Pairwise`, uses the *Pairwise* model that directly takes pairwise preferences as input, to generate the relative question and incorporate the observations. Thus, the user's relative feedback is incorporated into the model without any intermediate transformation, i.e., as an observation (i, B, A) when B is preferred over A.

The `Pairwise` method picks A exactly as in `Abs`, and for item B, inspired by the dueling bandit approach in [185], it picks the item with the largest probability of being preferred to item A. We instantiate the latter by selecting the item with the maximum noisy difference from item A (j^*):

$$\text{item B} = h^* = \arg\max_{h \in \mathcal{I}} \hat{y}_{ihj^*} \tag{5.7}$$

For the selection of item B, any question selection strategy besides TS illustrated here (except for MinT, MaxT), can be employed exactly as in Table 5.3, with the difference that the noisy *difference* distribution should be used.

Incorporating the 'Neither' Option. Preliminary experiments showed that when the method asks the user to give a relative preference on two items that he dislikes, forcing him/her to choose one could mislead the model about the users preferences. Thus, we adjusted all methods so that in such a case the user can specify that he likes neither. We implemented this by (i) incorporating two dislikes in `Abs Pos & Neg` and (ii) omitting the update in `Abs Pos` and `Pairwise`.

5.4 Experimental Setup

We now describe our overall empirical setup, used for the experiments in the next two sections.

Setting. One main difficulty of evaluating conversational recommenders is that it requires the ability to have access to user reactions to any possible system action. We address this requirement using generative user models. The first user model is constructed synthetically and is used to validate our model (Section 5.5). The second is instantiated from real user preferences, collected via a user study (Section 5.6).

All experiments consist of an offline and an online phase. During the *offline phase*, the model is presented with data where M users interact with N items. In the subsequent *online phase*, the model is used to interact with cold-start users, asking questions from the pool of the N offline items.

We varied the number of questions from 0 to 15 and report the model's performance after each question. In practice, recommendations could be given after fewer questions, could be integrated with initial recommendations, or could be spread out over several interactions with a given user.

Research Questions. Our experiments are designed to address the following key questions:

1. Can our model adapt to the user's preferences?
2. Does our model learn effectively under either absolute or relative feedback?
3. Which relative question method performs better?
4. Is absolute or relative feedback more effective?
5. Does the offline initialization step help?
6. Which question selection strategy is more effective?

Metric. To answer each of these questions, we need some measure of evaluating the effectiveness of our framework. Given that the goal of preference elicitation is a good recommendation list adhering to the user's preferences, we use *Average Precision@top* (*AP@top*) as our evaluation metric (Section 2.4), setting *top* to 10.

We report the average and 95% confidence intervals of AP@10 over all cold-start users. Higher value (closer to 1) of AP@10 implies better recommendation list. Results in additional metrics, such as ratio of correctly ranked pairs and mean reciprocal rank, were omitted as they showed similar trends.

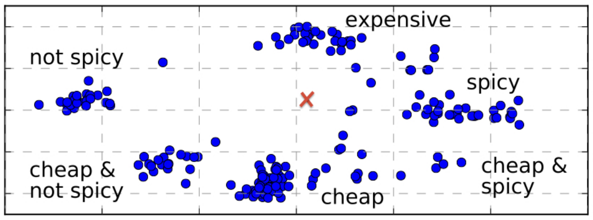
5.5 Learning Synthetic User Type Preferences

We begin our experiments with an intentionally simplistic example of restaurant recommendation, as real world high-dimensional data are difficult to visualize. The example is meant to demonstrate concepts of our model and to illustrate the effectiveness of our approach to unlearn initial prior beliefs and tailor recommendations to specific user types, answering the first question affirmatively.

In our framework, we first use observations to learn an offline embedding for users and items in the same low-d space. Here, we generated the offline observations by considering types of users and restaurants as follows:

Restaurant types	%	**User types**	%
expensive	15%	Like expensive	20%
cheap & spicy	5%	Like spicy	15%
cheap & not-spicy	10%	Like not-spicy	25%
only cheap	35%	Like cheap	30%
only not-spicy	15%	Like only not-spicy	5%
only spicy	20%	Like only spicy	5%

We generated $N = 200$ restaurants, and $M = 200$ users. For each offline user, according to their type, we sampled 10 items from their liked category as likes and 10 items from the rest of the categories as dislikes. We used this intentionally simple synthetic setup to evaluate various parameter choices, and we show results for $\sigma_1^2 = 10$, $\sigma_2^2 = 0.5$, $\epsilon = 0.1$. To allow visualization, we considered only two latent traits (d=2) for this example. We see that in the learned embedding, the first trait indicates spiciness, while the second the price.



In the same space, an embedding for users is also learned (not shown to avoid clutter). The average trait vector over all users is shown with a red cross. This becomes the initial trait vector for the cold-start user. Considering also the learned items' biases and the average user bias (not shown here), the system constructs an initial estimate of the noisy affinity distribution of the incoming user about all items.

Based on the offline observations, the learned prior for this affinity distribution favors user types which were popular in the offline data. The task of selecting restaurants for online users resembling the mean offline user is easy, as the prior already captures valuable information. As Fig. 5.1, *bottom right* panel shows, even with no questions, a close to perfect recommendation list can be given for *Liking not-spicy* users. Similar is the trend for the *Liking cheap* users (not shown).

In contrast, when the user is of a type that was rarely seen during the offline phase (e.g., expensive, shown in Fig. 5.1, *top right* panel), the online recommender has to collect observations that move away from more popular types. The trends for *Liking spicy*, *Liking only-spicy*, and *only not-spicy* user types are similar to the *Liking expensive*. For these types, the model (all four approaches) starts with AP@10 close to 0, but after every question asked, unlearns the initial wrong prior beliefs, and learns the specific user types' preferences. For the results reported, we considered 60 cold-start users of each type and used TS for the question selection.

All methods learn effectively across all user types, with minor differences (Fig. 5.1, *left*) answering the second question positively.

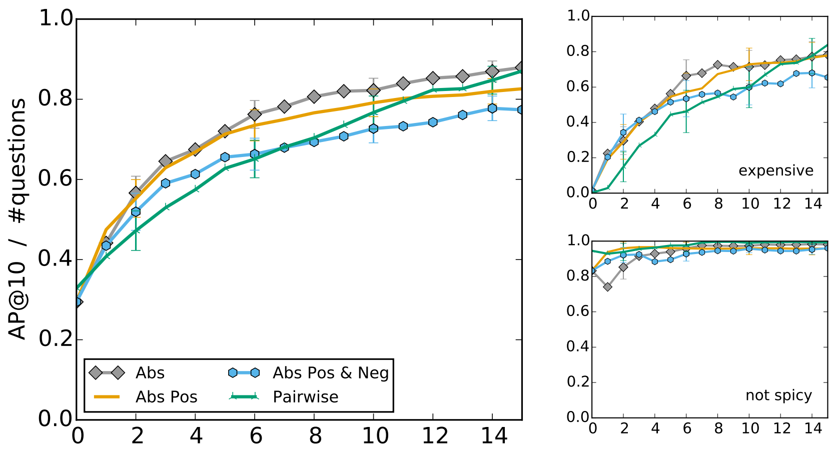


Figure 5.1: Results on synthetic restaurant data, across user types (*left*), and two of the user types (*right*).

5.6 Results on Real Data

Here, we turn our attention to a real-world setting to address all research questions in the context of 'where to dine in Cambridge, UK'. For the offline initialization of our framework, we use real users' online search behavior in a commercial search engine. We use the insights from Section 5.2 to design a user study that is used to collect real restaurant preferences for Cambridge. The collected responses serve as a basis for evaluating our online recommendation methods. We sketch a novel two-step approach to infer ratings for all restaurants, apart from those asked in the study. We extensively evaluate our choices for the recommendation pipeline.

5.6.1 Search Data for Offline Initialization

We start by describing the data which serve as our offline user-restaurants observations, based on which we learn the embedding used to warm start the question-asking phase. This data is obtained by identifying restaurant review pages for restaurants in Cambridge (UK) on a major restaurant review service provider. Next, we filter the query and click logs from a major commercial search engine to find (anonymous) cookies that mark a single PC accessing a sequence of these restaurant pages. Each cookie is taken to be a distinct user, and all visits on restaurant review pages are considered to be indicating the user liking the restaurant.[4]

In particular, taking logs from 26 March to 26 April 2015, we identified 3,549 cookies (users) who accessed at least one of the 512 distinct Cambridge restaurant review pages identified on the review service provider. This resulted in an index of Cambridge restaurants, each one visited by at least one user. Augmenting each of the restaurants with all known links and metadata associated with it in a proprietary restaurant index, and selecting a further three months back in each of those users' search histories, we recorded every interaction of these users with these restaurant links. During the total four-months search history of the users, we recorded interactions with 289 unique restaurants out of the 512 Cambridge restaurants. The total number of unique user-restaurant interactions recorded is 9330.

Thus, our offline data consists of $M = 3549$ users, $N = 289$ restaurants, and 9330 positive observations (1). To introduce dislikes (0) to the rating matrix as well, for every user i who has liked n_i^+ items, we sampled uniformly at random $n_i^- = \min(10, n_i^+)$ restaurants as dislikes.

To learn the offline embedding, we set the hyper-parameters to the combination that achieved the highest pairwise accuracy in the offline observations: $d = 4$ (i.e., 4 latent traits), $\sigma_1^2 = \sigma_2^2 = 10$, $\epsilon = 0.1$.

5.6.2 User Study as Basis for Online Evaluation

One of the issues of evaluating a conversational recommender is that one needs to know the user's ground truth on the space of all items (and questions). To obtain this, one needs to implement an interactive recommender asking questions to real users and receiving their responses. As an intermediate step, we conducted a user study and used the collected responses as ground truth for online users.

In the user study conducted, each participant filled in an anonymous web questionnaire about their preferences on a pool of restaurants in Cambridge, UK. The participants were asked "would you consider restaurant X for your next Friday night dinner?", labeling each restaurant with a binary label (yes/no). These responses comprise our ground truth.

For the pool of Cambridge restaurants we carefully selected ten restaurants, diverse in various features (as identified in Section 5.2). We recruited twenty eight individuals

[4] Though URL visitation is a weak positive signal, the experiments indicate it is a reasonable proxy for interest.

for the study. Given the anonymity of the questionnaire (in order to encourage truthfulness in the responses), demographic information was not recorded. However, the larger pool of individuals from which the participants were drawn (65 people working in a research lab) varied in factors such as age, job level, income, time spent in Cambridge.

Each participant was presented with the questionnaire about the same restaurants, but with varying order to avoid presentation bias. The participants were advised to visit the restaurant webpage when unfamiliar with the restaurant.

5.6.3 Obtaining Ground Truth

In our user study, we obtained restaurant labels for 10 out of the 289 Cambridge restaurants present in the offline data, for 28 participants. However, for reliable evaluation we need labels for the entire item space (rather than limiting our methods to ask about just these 10 restaurants). Therefore, we introduce an approach to fill in complete ground truth labels. Also, to increase the diversity of the user space, inspired by [110], we used bootstrapping to obtain 50 cold-start users based on the 28 participants' ground truth.

In particular, for each cold-start user:

1. Randomly sample one of the 28 participants.
2. Observe the sampled user's labels on the pool of 10 restaurants asked in the user study.
3. Infer user's traits $\mathbf{u}_i$ (prior= learned embedding in 5.6.1).
4. *Sample* $\tilde{\mathbf{u}}_i \sim \mathbf{u}_i$. Set this to be the new prior of $\mathbf{u}_i$.
5. With this prior, infer the ratings $\mathbf{r}_i$ distribution.
6. *Sample* ratings from their distribution $\tilde{\mathbf{r}}_i \sim \mathbf{r}_i$.

In this way, for each bootstrapped user we obtain a complete rating list for all 289 restaurants that is consistent with the user study labels of some user, yet is perturbed to account for variety in real user populations.

As far as we are aware, this approach for filling in the missing ratings is novel. It gives us ground truth as close to real as possible given the resources available. Alternatives would be exhaustive labels (not feasible for our study), or rejection/importance sampling (only effective when leveraging logged exploration data from large-scale systems, e.g [104]).

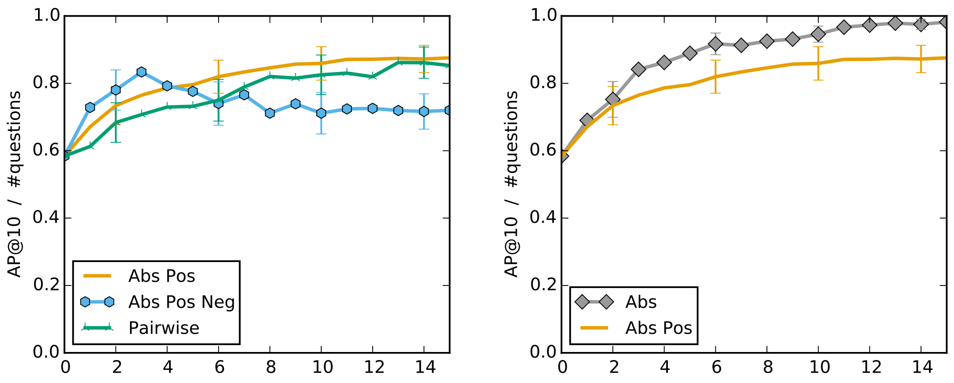


Figure 5.2: Differences between relative feedback models (*left*); and comparing absolute and relative feedback (*right*).

5.6.4 Results

Having obtained the offline embedding and the online users' ground truth for all items, we now present our experiments on a real restaurant recommendation scenario with the focus of answering the remaining research questions. Each of these questions investigates a separate component of our continuously learning recommender system.

Which method for relative questions is better?

Recall that we proposed three approaches for modeling relative feedback. The results of the three methods' comparison are shown in the *left* panel of Figure 5.2. These results were obtained using TS for question selection and start from the offline embedding. We see no significant difference among the methods during the first few questions. After only 2 questions, all methods significantly improve over the initial performance of .584 to respectively .734 (`Abs Pos`), .780 (`Abs Pos & Neg`), and .684 (`Pairwise`). However, as we ask more questions, `Abs Pos & Neg` forces negative observations on liked restaurants, thus causing the method to degrade the ranking. Overall, `Abs Pos` is the most effective method for relative questions.

Are absolute or relative questions better?

To answer this question, we compare the performance of the absolute-question asking method (`Abs`) with the best relative-question asking method (`Abs Pos`). The results are

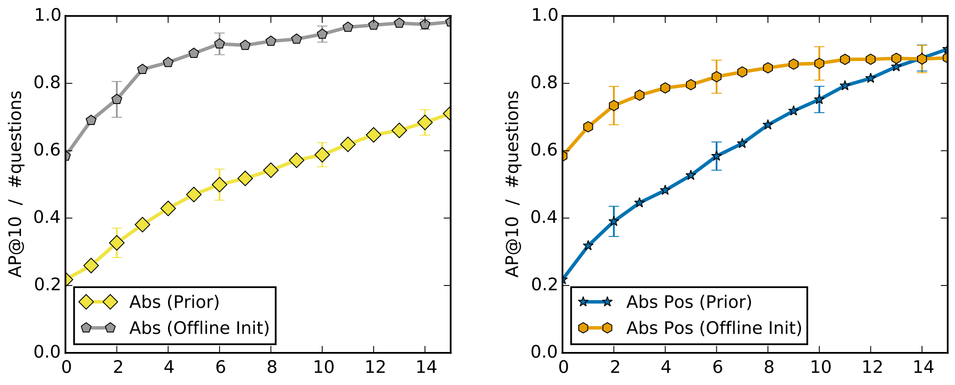


Figure 5.3: Impact of offline initialization on performance for absolute (`Abs`, *left*) and relative feedback (`Abs Pos`, *right*).

shown in the *right* panel of Figure 5.2. Until 2 questions, both methods have almost identical performance. But, after 5 questions, `Abs` performs significantly better than the relative feedback method, and achieves close to perfect performance after 15 questions ($AP@10 = .975$). We hypothesize that this result can be explained by the fact that our offline embedding favored absolute feedback.

Although our result shows that very high accuracy can be achieved when users provide accurate feedback on absolute questions, in practice this may not always be possible. Psychological effects such as anchor bias [89] can lead users to implicitly compare items, lowering the quality of absolute feedback. When this is the case, our result shows that high performance can be achieved with relative feedback as well.

Does offline initialization help?

Next, we investigate the impact of model initialization, i.e., initializing the online recommender with an initial *offline embedding*, compared to starting from a generic prior. We present the results of this comparison in Figure 5.3 both for the absolute (`Abs`) and the best relative feedback model (`Abs Pos`), in the left and right panel correspondingly.

Our hypothesis is that the offline embedding learned from the weakly labeled data of a search log captures sufficient information to helpfully constrain continued learning, even if it does not exactly match the structure that underlies online users. Indeed, Figure 5.3 demonstrates great performance improvements when initializing the models from this embedding over generic prior initialization. By placing the new users as the average offline user, performance increases from .217 to .584, even without asking any questions. As the recommender collects more responses, performance continues to improve in both cases.

One observation is that the uninitialized system can ultimately achieve high performance, and for `Abs Pos (Prior)` even pass the offline initialized system. However, this is only achieved after many questions (here: 14). The phenomenon is nevertheless interesting, as it may point to a bias-variance trade-off. Starting from the generic prior allows the system to eventually perfectly fit a given user's preferences, however, learning takes a long time because there is no initial structure to constrain learning outcomes. We conclude that using offline initialization is highly beneficial.

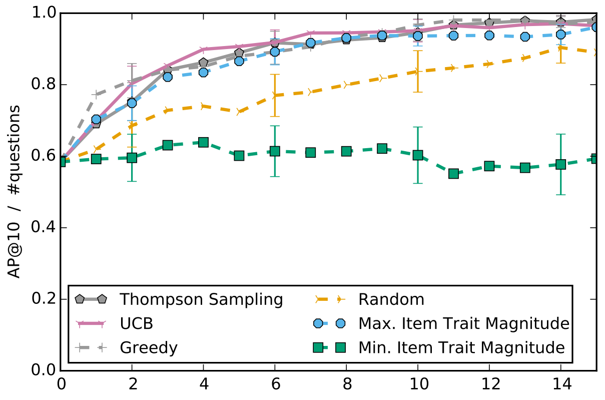
Which question selection strategy is best?

In Figure 5.4 we report the comparison results of the various question selection strategies of Section 5.3.3 (except `MaxV` whose performance almost coincides with `UCB`), for the `Abs` and `Abs Pos` methods initialized with the offline embedding.

For the `Abs` model (Figure 5.4, *top*), we observe that: (i) as expected, lowest AP@10 is achieved by `MinT`, (ii) `Random` learns slowly, likely because it fails to focus on more promising items for effective learning, and (iii) all remaining strategies perform equally well. We hypothesize that `Greedy` performs well thanks to the offline embedding, along with the online updating of all parameters after each response.

Turning to `Abs Pos` (Figure 5.4, *bottom*), the best performing strategies are those that encourage more diversity across the questions of the interactive session, namely the bandit-based ones and `Random`. `Greedy` and `MaxT` are the worst performing ones, following `MinT`. Our insight why this is the case is that they tend to select questions *A vs B*, followed by *A vs C*, etc. when A is preferred. Given that there is no construction encouraging B and C to be diverse (such as sampling or taking into account uncertainties), the questions focus on comparing parts of the embedding which are similar across questions, thus preventing truly effective learning.

Overall, we find that the bandit-inspired strategies perform the most robustly, achieving top performance across models. In the classic bandit setting, these approaches systematically balance the need to explore new solutions with the need to reap the rewards of what has already been learned. Here, we find that similar principles allow these strategies to collect user feedback that balances the need to not discard any areas of the latent space prematurely with the need to focus questions on the most promising areas. This insight is important because it shows that bandit-based question selection strategies can lead to benefits that go beyond the typical bandit problem.



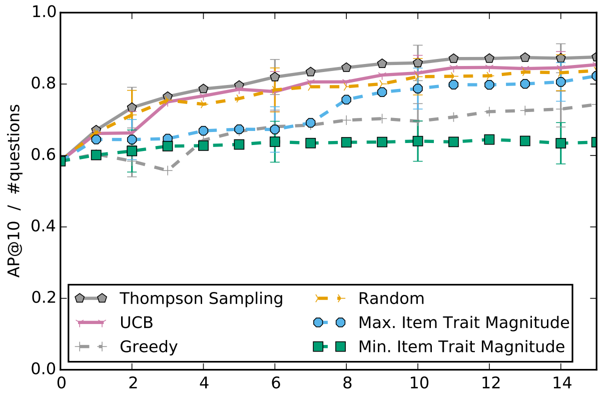


Figure 5.4: Performance of question selection strategies for: *Top*: absolute (`Abs`); *Bottom*: relative (`Abs Pos`) models.

Discussion

All our results show that our methods enable effective learning from interactions with their users, under either feedback type, answering positively the first two questions.

We show substantial recommendation performance improvements over the performance we would get without adapting to the user; by 25% after only 2 questions.

Although our underlying model can be augmented with external features [160], one key advantage is it does not need them. It learns online both the user's and the items' latent traits. Together with [112] and [184], our findings corroborate the effectiveness of latent-feature interactive recommenders.

A novel insight is that taking into account the uncertainties in the learning of *both* the item and the user embedding, we can adapt to the specific user's preferences, under a certain context. This answers to the question posed by [184] and is key for allowing the system to learn both the user's profile and the questions' effectiveness in a contextual manner.

We have demonstrated effective learning from feedback present in many interactive settings. Thus, our approach is a good candidate for online learning across domains.

5.7 Related Work

The presented work in this chapter builds on work in multiple active research areas. We review recent work in the most closely related ones.

Online Recommenders. Although most works on recommendation have focused on offline recommenders, a recent line of work has posed the problem in a *contextual bandit* formulation [104, 165, 169], where items are seen as *arms*, users as *contexts*, and the goal is to *explore* the arm space in order to *exploit* the best performing arm for a given context (section 2.1). Most work in this area relies on the availability of user/item features and assumes that the reward of an arm for a given context is a linear/logistic function of the concatenated feature of arm-context [104, 105, 165]. [169] uses Gaussian process kernel functions on top of contextual bandits to allow effective feedback sharing among similar contexts and arms.

Using the collaborative filtering (CF) point of view, [30] introduced an ϵ-greedy online user-user *neighbor-based* CF method. The first *latent-factor* online recommender was introduced in [184], which uses bandit strategies on top of probabilistic matrix factorization (PMF). However, while [184] fixes the item latent factors to those learned offline, formulating the problem as a *contextual linear bandit*, our method *fully online* lears *all* user and item parameters, including the biases; [184] can be seen as a special case of our framework. In [91], the authors extend Thompson Sampling for PMF with a Kalman filter to track changing preferences over time. This work is orthogonal to ours.

Preference Elicitation. The problem of eliciting user feedback has long been of interest for a variety of tasks (e.g [52]). To elicit the preferences of an existing or new user (cold-start), a range of methods have been proposed, varying from interview-based strategies (e.g [162]), to asking users to rate some items, to active learning [145], entropy minimization [149], picture-based [123], and explore-exploit strategies on top of a latent factor model [184]. Our work is the first to elicit users' preferences by utilizing either absolute or relative feedback in a fully online setting.

Interactive Recommenders. Many works (critique-based [40], constraint-based [64], dialog, utility-based recommenders [113]) have emphasized the importance of interactivity in recommenders so that the user has a more active role over the recommendations. However, these works rely on prior modeling of the items' features, preventing the flexibility in adaptation to a different domain; thus a comparison with them is out of the scope of this work.

In contrast, our work, in the same spirit as a recent line of work [112, 75] initiated by [184], learns online the *latent factors* from PMF and uses *these* to do interactive preference elicitation. These methods, although close in principle, have significant differences from our work. Briefly, in [112], the authors focus on set-based feedback and propose a progressive building of the user's latent factor in each question. In contrast, our work focuses on absolute and relative feedback on (pairs of) items, and uses a preference elicitation phase fully integrated with the online updates of the PMF model. The method of [75] focuses on choice-based feedback and updates online only the user's latent factor. Neither of [75, 112] balance the explore-exploit trade-off.

5.8 Summary

In this chapter, we proposed a novel view of recommendation as an interactive process. Like human recommenders, we envision recommender systems that can converse with new users to learn to know their preferences.

We develop such a conversational recommender, using restaurant recommendation as our motivating example. We started by examining restaurant related queries issued in a commercial search engine. Using these insights, we conducted a user study to elicit ground truth for evaluating our system. We proposed a conversational recommender model that is theoretically well anchored in probabilistic matrix factorization models [118, 160]. We showed how such models can be extended to support continuous learning. We empirically evaluated our approach using the ground truth based on real dining preferences elicited through our user study.

Our results have important implications for the development of conversational recommender systems. First, we demonstrated that best performance can be achieved with absolute questions. However, even in settings where only relative feedback is available, effective learning is possible. Second, we proposed a systematic approach to initializing conversational recommenders with an offline learned embedding, boosting greatly the performance even when only weakly supervised data is available. Third, we identified question selection strategies that can elicit feedback for very effective learning. Together, these insights pave the way towards conversational recommenders.

A promising future direction is to extend conversational recommenders to use reinforcement learning approaches, for capturing longer-term dependencies [110]. The modular structure of our framework allows various choices in a plug-and-play-manner, considering different feedback types, underlying probabilistic models etc., with the goal of building a suite of conversational recommenders for a variety of settings.

Chapter 6

Recommendation as Collaboratively Learning to Interact

Which interactive recommendation systems are the most effective?

Learning to interact with users and discover their preferences is central in most web applications, with recommender systems being a notable example. From such a perspective, merging interactive learning algorithms with recommendation models is natural. While recent literature has explored the idea of combining collaborative filtering approaches with bandit techniques, there exist two limitations: (1) they usually consider Gaussian rewards, which are not suitable for implicit feedback data powering most recommender systems, and (2) they are restricted to the one-item recommendation setting while typically a list of recommendations is given.

In this chapter, to address these limitations, apart from *Gaussian rewards* we also consider *Bernoulli rewards*, the latter being suitable for dyadic data. Also, we consider two *user click models*: the one-item click/no-click model, and the cascade click model which is suitable for top-K recommendations. For these settings, we propose novel machine learning algorithms that learn to interact with users by learning the underlying parameters collaboratively across users and items. We provide an extensive empirical study, which is the first to illustrate all pairwise empirical comparisons across different interactive learning algorithms for recommendation.

Our experiments demonstrate that when the number of users and items is large, propagating the feedback across users and items while learning latent features is the most effective approach for systems to learn to interact with the users.

6.1 Introduction

Learning to interact with users is at the core of many web applications, ranging from search engines, to recommender systems, and more [85]. The reason is that these systems exhibit the user-system interaction loop: every time a user uses the system, the system has decided to show a list of items to the user, from which the user selects zero, one, or more items; then, the system gets the user's feedback (in terms of clicks, time spent, and other signals) to update its model of the user's preferences—which might affect the system's future decisions on this and other users.

This inherently interactive nature of web systems creates the need for machine learning models that *learn* how to interact with users over time; instead of static models trained based on offline data, that are periodically retrained to incorporate newly acquired information. Such interactive learning models are largely composed of two parts: which technique is used to balance the need for exploring user preferences vs. exploiting what has been learned so far, and what are the underlying model assumptions for the user reward?

In this chapter, inspired by recent works which pose recommendation as an interactive learning problem [104, 106, 184, 91], we build on two good ideas, corresponding to each of these two parts. First, given that the recommender system sequentially learns about its users and items from repeated interactions, we make use of the construction of multi-armed bandits [152]—we particularly focus on Thompson Sampling thanks to its good empirical guarantees [38]. Second, the collaborative filtering principle, i.e., similar users tend to like different items similarly, is a powerful tool which can be used to inform the underlying interactive learning models. By merging the two ideas, one can develop algorithms which learn to effectively interact with users, by propagating the user feedback across users and items. The construction, although illustrated in the context of recommender systems, can be used in other web systems where personalization plays a key factor, ranging from medical interventions, to personalized search engines, and more.

Particularly, our contributions are three-fold:

1. As most user interaction data are implicit and are an important signal for learning

recommendations [81, 87], we develop new algorithms well-suited for Bernoulli rewards (Sections 6.4.1, 6.4.2).

2. Establishing the connection among one-item and cascade list recommendation algorithms (Section 6.3), we offer the novel construction of *collaborative cascade ranking bandits* (Section 6.4.2).
3. Empirical evaluation carefully considers both the one-item and the cascade list setup, under both Gaussian and Bernoulli rewards; as far as we know, this is the first study making all such pairwise comparisons (Section 6.5).

Our extensive experiments show that for both the one-item and cascade list setup, when the number of users and items in the system is large, (1) collaborative bandits outperform running individual bandits per user; and (2) collaboratively learning across users indeed outperforms the contextual linear and clustering principle in the interactive learning setting.

6.2 Key Concepts & Overview

We start with a discussion of some useful concepts.

6.2.1 Collaborative Filtering

Let M be the number of users, N the number of items, and $R \in \mathbb{R}^{M \times N}$ the rating matrix of users' ratings on items. According to the latent-factor collaborative filtering assumption (Section 2.2), the matrix can be approximated by the inner product of two low dimensional latent factor matrices: $U \in \mathbb{R}^{M \times d}$ and $V \in \mathbb{R}^{N \times d}$, which represent the latent features of the users and items respectively [148, 147]

6.2.2 Recommendation as Learning to Interact

Recommender systems sequentially learn the users' preferences based on repeated interactions with them. Let's assume that at each interaction round, a user, randomly drawn from the user population, comes to the system. Then, the system-user interaction model is characterized by two steps:

- **step 1:** the decision step, where the system decides to show an item (or list) to the user, on which the user gives feedback (reward) explicitly or implicitly; and

- **step 2:** the model update step, where the system updates its parameters using the user-provided feedback.

The system's goal is to select items so that maximum cumulative reward (or minimum cumulative regret) over the total T interaction rounds is achieved.

Given the sequential nature of the interactions, every user can be seen as a *multi-armed bandit* [152]—that is, for every user who interacts with the system, the parameters underlying the user's reward model have to be sequentially learned. The *arms* or *actions* of the user-bandit are the candidate items for recommendation. In the rest of the chapter, we will use the terms users/bandits and items/arms interchangeably. For a user $i \in \mathcal{U}$, let $\mathcal{L}_i$ be the set of active arms, which are the candidate items for recommendation (noting that the cardinality of $\mathcal{L}_i$ can be less than N). The reward of item j for a user i is denoted by r_{ij} and captures the feedback (e.g., click/no-click, buy/not-buy, rating) user i gives on item j. Rather than the reward matrix $R = [r_{ij}]$ being arbitrary, we will assume that R has a parametric form, and specific rewards r_{ij} depend on the user i, the item j, and the unknown parameters $\boldsymbol{\theta}^*$ (such an assumption will be empirically evaluated in Section 6.5).

6.2.3 User Reward Models

We assume that the rewards of the arms per bandit are stochastic [56]. This allows for a flexible setting where user feedback is noisy, as is the case in various recommendation scenarios, including news and content recommendation [7]. We consider the following two types of rewards:

- **Gaussian Rewards:** the reward of item j for user i is modeled with a Gaussian distribution $r_{ij} \sim \mathcal{N}(\mu^*_{ij}, \sigma^2_{ij})$, $r_{ij} = \mu^*_{ij} + \epsilon_{ij}$, where $\epsilon_{ij} \sim \mathcal{N}(0, \sigma^2_{ij})$ is the noise and μ^*_{ij} is the average rating of user i on j.
- **Bernoulli Rewards:** suited for settings where a user gives binary feedback (explicit, e.g., thumbs up/down on a song, or implicit e.g., click/skip on a news article). Let $\pi^*_{ij} \in [0, 1]$ be the probability that user i likes an item j. Then, we model the reward of user i on j with a Bernoulli distribution $r_{ij} = \text{Ber}(\pi^*_{ij})$, i.e., $r_{ij} = 1$ with probability π^*_{ij} and $r_{ij} = 0$ with probability $1 - \pi^*_{ij}$.

In both reward models, the reward of item $j \in \mathcal{L}_i$ for every user i is modeled based on a true parameter θ^*_{ij} (π^*_{ij} for Bernoulli rewards or μ^*_{ij} for Gaussian rewards). If we knew the true value of θ^*_{ij}, at every round the system would pick for incoming user i the item that maximizes the expected reward, $\max_j \mathbb{E}(r_{ij}|i, j, \theta^*_{ij})$.

6.2.4 Random Probability Matching Principle

Since we do not know the true value of the underlying parameters $\boldsymbol{\theta}^*$, we maintain a distribution over the parameters $P(\boldsymbol{\theta})$. If we wanted to maximize the immediate reward in a pure exploit setting, one would choose for user i the item that maximizes $\mathbb{E}[r_{ij}|i, j] = \int \mathbb{E}[r_{ij}|i, j, \theta_{ij}]P(\theta_{ij}|\cdot)d\theta_{ij}$. However, to balance the explore-exploit (EE) trade-off, Thompson Sampling (TS) [167] uses the probability matching heuristic, and chooses for bandit i an action j according to its probability of being optimal [38], i.e., with probability:

$$\int \mathbb{1}\left[\mathbb{E}[r_{ij}|i, j, \theta_{ij}] = \max_{j' \in \mathcal{L}_i} \mathbb{E}[r_{ij'}|i, j', \theta_{ij'}]\right] P(\theta_{ij}|\cdot)d\theta_{ij},$$

where $\mathbb{1}[\cdot]$ is the indicator function, the $\mathbb{E}$ inside the integral denotes expectation over the stochastic nature of the rewards, and the integral denotes expectation over the parameter distribution. The integral can be approximated by drawing for every arm j a sample from the corresponding posterior $P(\theta_{ij}|\cdot)$. TS chooses the arm with the largest drawn posterior sample.

Various EE strategies such as UCB [18], ϵ-greedy or TS have been used in the different works posing recommendation as a learning to interact problem [104, 106, 184, 91]. Throughout this chapter, we use TS as it can be efficiently implemented and it has been shown to have competitive empirical performance [38].[1]

6.2.5 Click Models

- **One/no-click:** a single item is presented to the user and the user decides to click or skip (or to explicitly like or dislike). We interchangeably refer to this as *one-item.*
- **Cascade Model:** originally introduced to study web search behavior, it models the interaction of the user with a top-K list [54]. It assumes that the user examines the list top to bottom, and clicks one item as soon as they find an interesting one.

[1] Our goal is *not* to compare TS with other EE strategies, or to compare alternative ways of posterior sampling; rather, keeping TS as the EE strategy of our choice, we aim to compare the interactive learning recommendation algorithms.

The user decides whether to click on each item before moving to the next, and the click probability on each item is independent from the rest. Thus, if a user clicks on item j at position k (j_k), they dislike the items on the above $k-1$ positions, and ignore the items from $k+1$ onward. The probability of clicking item j_k is $c_{j_k} = r_{j_k} \prod_{l=1}^{k-1}(1 - r_{j_l})$, where r_j is the probability of clicking item j, and $1 - r_j$ is the skipping probability.

6.3 Learning to Interact with Users

Recommender systems that *learn* to interact with users balance the need to learn new information about the user (*explore*) while also focusing on what has already been learned about their preferences (*exploit*). They achieve this for a user i in the following way. They start with a prior distribution over the latent parameters of the reward distribution $p^0(\theta_{ij})$ for every item j. Typically, a conjugate prior of the likelihood $p(\mathcal{D}|\theta_{ij}, \cdot)$ is used, where $\mathcal{D}$ is the collection of the thus far observed rewards, so that the posterior distribution $p(\theta_{ij}|\mathcal{D}, \cdot) \propto p(\mathcal{D}|\theta_{ij}, \cdot)p^0(\theta_{ij})$ is of the same form as the prior. Next, for every item j the system gets a sample from the posterior $\tilde{\theta}_{ij} \sim P(\theta_{ij}|\mathcal{D}, \cdot)$, and shows the item that results in the largest reward $\tilde{r}_{ij}$, estimated as a function of $\tilde{\theta}_{ij}$. The user gives feedback on the shown item $\hat{j}$, and based on this feedback the system updates its belief about the underlying parameters of the reward distribution. In the next interaction round the updated posterior distribution becomes the new prior. The wider the posterior distribution, the more the system explores. As more feedback is collected, the posterior distribution $P(\theta_{ij}|\mathcal{D}, \cdot)$ becomes more peaked and shifts to better capture the mean of the true underlying parameters θ^*_{ij}—thus, the system exploits more.

One/no-click vs. Cascade list. Algorithms 3 and 4 show the discussed procedure for the one/no-click model and the cascade list model respectively. Comparing the two algorithms, it is easy to establish the connection among learning to interact algorithms for one-item and cascade list recommendation. The cascade list recommendation algorithms differ from the click/no-click ones in two aspects: (i) they select the top K items instead of just the top one, and (ii) the model updates its parameters based on skips/clicks for all items up until and including the clicked one; the model does not perform any update for the items after the click. Later in the chapter, we will exploit this connection to create novel learning to interact algorithms for the cascade model.

Algorithm 3 TS-based One-Item Recommendation

```
Require: hyper-parameters of the prior $P^0(\theta)$.
 1: for round $t = 1, \dots, T$ do
 2:     User $i \sim$ Uniform$(1, M)$ comes to the system.
 3:     for $j = 1, \dots, |\mathcal{L}_i|$ do
 4:         Draw $\tilde{\theta}_{ij} \sim$ Posterior distribution $P(\theta_{ij}|\mathcal{D}, \cdot)$.
 5:     end for
 6:     $\forall j$: estimate reward as a function of the sampled parameters $\tilde{r}_{ij} = f(\tilde{\theta}_{ij}, \cdot)$.
 7:     Show the item $\hat{j} = \arg\max_j \tilde{r}_{ij}$.
 8:     Observe reward $r_{i\hat{j}}$ : (i) for Bernoulli: $\sim$ Ber$(\theta^*_{i\hat{j}})$, (ii) for Gaussian: $\sim \mathcal{N}(\theta^*_{i\hat{j}}, \sigma^2_{ij})$.
 9:     Update parameters $\theta$.
10: end for
```

Algorithm 4 TS-based Cascade List Recommendation

```
Require: hyper-parameters of the prior $P^0(\theta)$.
 1: for round $t = 1, \dots, T$ do
 2:     User $i \sim$ Uniform$(1, M)$ comes to the system.
 3:     for $j = 1, \dots, |\mathcal{L}_i|$ do
 4:         Draw $\tilde{\theta}_{ij} \sim$ Posterior distribution $P(\theta_{ij}|\mathcal{D}, \cdot)$.
 5:     end for
 6:     $\forall j$: estimate $\tilde{r}_{ij} = f(\tilde{\theta}_{ij}, \cdot)$, sort items based on $\tilde{r}_{ij}$, and show the top $K$ items.
 7:     Observe feedback on position $C_t$ (at most 1 click).
 8:     for $l = 0, \dots, \min(C_t, K)$ do
 9:         Update parameters for item $\hat{j}$ at position $l$.
10:     end for
11: end for
```

6.3.1 Parametric Assumptions for User Rewards

The various learning to interact recommendation algorithms differ only in their parametric assumptions for the reward distribution; specifically in the input prior distribution (line 0), the posterior distribution (line 4), the definition of reward as a function of the parameters θ (line 6), and the way they update the parameters (line 9). Some parametric assumptions are:

1. *Independent* [38]: The reward parameters of the items j, j' of a given user i are uncorrelated, i.e., $\theta^*_{ij} \neq \theta^*_{ij'}$. Also, the parameters for every user i are independent from the parameters of every other user $i' \in \mathcal{U}$.
2. *Contextual Linear* [104, 105]: Every arm j is represented by a feature vector $\mathbf{x}_j \in \mathbb{R}^d$ representing the context. For a user i the rewards of the different arms are correlated via a common parameter vector $\boldsymbol{\theta}^*_i \in \mathbb{R}^d$ capturing how important each dimension of the context is: $\mathbb{E}[r_{ij}] = h(\boldsymbol{\theta}^{*T}_i \mathbf{x}_j)$, where for Gaussian rewards

h is the identity function, while for Bernoulli rewards it is the sigmoid function $\sigma(\cdot) = \frac{1}{1+\exp(-\cdot)}$.

3. *Contextual Clustering*[125, 68, 69, 93]: This case assumes that there exist clusters of the user population which have similar rating behaviors over the items, and contextual features $\mathbf{x}_j \in \mathbb{R}^d$ are available for each item j. All users who belong to the same cluster c will share the same reward parameter vector $\boldsymbol{\theta}_c^* \in \mathbb{R}^d$: $\mathbb{E}[r_{ij}] = h(\boldsymbol{\theta}_c^{*T}\mathbf{x}_j)$. The parameters of cluster c are independent from those of any other cluster c'.
4. *Low Rank CF* [51, 91, 184]: Using the CF view and particularly the low rank assumption (Section 6.2.1), both users and items can be represented by latent feature vectors. Concretely, considering $\forall i \in \mathcal{U}\ \mathbf{r}_i \in \mathbb{R}^N$ the vector of true underlying rewards, we assume the reward matrix $R \in \mathbb{R}^{M\times N}$ is drawn from a parameter matrix Θ^* which is a function of a low rank matrix. Thus, if $\mathbf{u}_i^*$ denotes the i-th row of the true latent user matrix U^* and $\mathbf{v}_j^*$ the j-th row of the true underlying latent item matrix V^*, $\mathbb{E}[r_{ij}] = h(\mathbf{u}_i^{*T}\mathbf{v}_j^*)$.

Note that with the independent assumption, if $\forall i \in \mathcal{U}, |\mathcal{L}_i| = N$, $M \times N$ parameters need to be learned. In contrast, the other parametric models do parameter sharing, coupling the items (and the users/clusters). Contextual linear and clustering models assume that contextual features are available and informative, which is not always true. Also, while both clustering and CF models share parameters among users with similar rating patterns, the former capture coarser user preferences represented by the clusters, whereas the latter can collaboratively learn finer grained latent preferences.

6.3.2 Existing Algorithms and Outline

For *One-Item Recommendation:* As far as we know, the independent assumption has been employed only within the context of a *single* multi-armed bandit [38], not as a baseline in multiple user-bandits for recommender systems—we have derived `Gauss/ Bernoulli Independent TS`. Using the contextual linear principle has led to the pioneering work for applying bandits on news recommendation, resulting in `LinTS` and `LogTS` for Gaussian and Bernoulli rewards respectively [104, 105]. The contextual clustering principle has been applied for Gaussian rewards, resulting in `CluLinTS` (in Section 6.5 we experiment with the variant of [125], but other sophisticated variants exist [68, 69, 93])—we have also devised `CluLogTS` for Bernoulli rewards. The CF low rank principle has been applied on interactive recommenders for Gaussian rewards, giving rise to `Gauss Low Rank TS` [91] (also, the CF principle has been applied using co-clustering instead of matrix factorization in [106]). In Section 6.4.1, we will offer collaborative low rank bandits for Bernoulli rewards as well. Its predecessor was `Gauss/ Bernoulli ICF`, standing for Interactive Collaborative Filtering, where the items' latent factors V are pre-learned and used as contextual features, allowing to pose the problem in a contextual linear setting [184].

For the *Cascade setup*: The independent assumption has been applied, resulting to `Gauss/Bernoulli Independent Cascade TS` [96]. Also, the contextual linear principle has been used, leading to `CascadeLinTS` [186] for Gaussian feedback—we have also devised `CascadeLogTS` for Bernoulli rewards. On the one hand, these methods are suitable for large-scale recommendation data, since the independent assumption does not scale well when the number of items candidate for recommendation is large (as is usually the case) [186]. On the other hand, they suffer from the following drawbacks: they rely on contextual features, and they do not propagate feedback among users, thus neglecting that similar users tend to like the various items similarly. In Section 6.4.2, we will apply the low rank CF assumption, giving rise to the novel construction of *collaborative low rank cascade bandits*. Also, we have devised `Gauss/ Bernoulli ICF Cascade TS`; specifically we have formulated the problem as a separate `CascadeLinTS` or `CascadeLogTS` per user, and used as contextual features the pre-learned latent attributes of the items.[2]

The parameter updates of the existing and novel interactive recommendation algorithms are in the Appendix A.1.

6.4 Learning to *Collaboratively* Interact with Users: Proposed Algorithms

In this section we proceed with our novel contributions on learning to interact with users by *collaboratively* learning parameters across them.

In our formulations, we build on two good ideas: (1) collaborative filtering and (2) interactive learning. The merge of these two ideas is key for efficiently propagating feedback among users, while at the same time learning the low dimensional embeddings of users and items on the same underlying latent space. The construction is suited for

[2] We omit the development of clustering cascade bandits, as early experiments showed the value of low rank.

scenarios where contextual features are not explicitly given, and the fine-grained similar rating patterns are to be discovered collaboratively.

Although the combination of the two ideas has been explored in [51, 91, 106], we propose two technical developments: (1) We develop collaborative *bernoulli* bandits referred to as `Bernoulli Low Rank TS`, considering the popular logistic matrix factorization model [87] which is successful for modeling click/no-click data, which are widely available in recommender systems. This is useful as most recommendation algorithms for top-K recommendation are powered from implicit type data [81]. (2) Having realized the connection among the one-item and cascade setup (Section 6.3), we develop collaborative interactive learning algorithms for the cascade setting both for Gaussian and Bernoulli user reward models.

6.4.1 One-Item Bernoulli Collaborative Bandits

In this setting, we use the low rank CF assumption to couple the probabilities $\{\pi_{ij}\}$ across users $\{i\}_{i=1}^{M}$ and items $\{j\}_{j=1}^{N}$. We assume that every entry π_{ij} of the matrix $\Pi \in \mathbb{R}^{M\times N}$ (i.e., the probabilities of how much each user likes every item) will be estimated by the sigmoid function of the inner product of $\mathbf{u}_i$ and $\mathbf{v}_j$. The sigmoid function is used to map the entries to the range of [0, 1] as they represent probabilities. Particularly, `Bernoulli Low Rank TS` assumes that $\mathbb{E}[r_{ij}] = \sigma(\mathbf{u}_i^T\mathbf{v}_j) = 1/(1+\exp(-\mathbf{u}_i^T\mathbf{v}_j))$. The model considered for parameter estimation follows the modeling assumptions of probabilistic logistic matrix factorization [87] and is also related to [58]. At step t:

$$\forall i: \quad \mathbf{u}_i^t \sim \mathcal{N}(\hat{\mathbf{u}}_i^{(t-1)}, S_{\mathbf{u}_i}^{t-1}) \quad \forall j: \quad \mathbf{v}_j^t \sim \mathcal{N}(\hat{\mathbf{v}}_j^{(t-1)}, S_{\mathbf{v}_j}^{t-1})$$
$$\forall ij: \ \pi_{ij}^t = \sigma(\mathbf{u}_i^{t^T}\mathbf{v}_j^t), \ r_{ij}^t \sim \text{Bernoulli}(\pi_{ij}^t)$$

The log of the posterior distribution over the user and the item latent features U, V is:

$$\begin{aligned}
&\sum_i \sum_{\hat{j}} r_{i\hat{j}}^t \ln \pi_{i\hat{j}}^t + (1-r_{i\hat{j}}^t)\ln(1-\pi_{i\hat{j}}^t) \\
&-\frac{1}{2}\sum_{i=1}^{M}(\mathbf{u}_i^t - \hat{\mathbf{u}}_i^{(t-1)})^T S_{\mathbf{u}_i}^{(t-1)^{-1}}(\mathbf{u}_i^t - \hat{\mathbf{u}}_i^{(t-1)}) \\
&-\frac{1}{2}\sum_{j=1}^{N}(\mathbf{v}_j^t - \hat{\mathbf{v}}_j^{(t-1)})^T S_{\mathbf{v}_j}^{(t-1)^{-1}}(\mathbf{v}_j^t - \hat{\mathbf{v}}_j^{(t-1)}).
\end{aligned} \tag{6.1}$$

If we fix the latent item features V and consider a single user i, Equation (6.1) reduces to a logistic regression problem with parameter vector $\mathbf{u}_i$. Given that exact Bayesian inference for logistic regression is intractable, we use the Laplace approximation to approximate the conditional posterior of $\mathbf{u}_i$ with a Gaussian distribution $q(\mathbf{u_i}) = \mathcal{N}(\mathbf{u}_i|\hat{\mathbf{u}}_i, S_{\mathbf{u}_i}, \cdot)$, where $\hat{\mathbf{u}}_i = \mathbf{u}_i^{\mathrm{MAP}}$ is the mode of the posterior and $S_{\mathbf{u}_i}$ is the Hessian matrix of second derivatives of the negative log posterior [21]. We compute the mode of the posterior by minimizing the negative log posterior with online gradient descent $\hat{\mathbf{u}}_i^{t+1} \leftarrow \hat{\mathbf{u}}_i^t - \eta_t \nabla_{\mathbf{u}_i}^t$:

$$\nabla_{\mathbf{u}_i}^t = S_{\mathbf{u}_i}^{(t-1)^{-1}} (\hat{\mathbf{u}}_i^t - \hat{\mathbf{u}}_i^{t-1}) + (\pi_{i\hat{\jmath}}^t - r_{i\hat{\jmath}}^t)\mathbf{v}_{\hat{\jmath}}^t,$$

where the step size follows the Adagrad rule $\eta_t = 1/\sqrt{\sum_{\tau=0}^{t-1} \nabla \mathbf{u}_{i,\tau}^2}$. The inverse of $S_{\mathbf{u}_i}$, denoted as $S_{\mathbf{u}_i}^{-1}$, is constructed online as:

$$S_{\mathbf{u}_i}^{t+1^{-1}} = S_{\mathbf{u}_i}^{t^{-1}} + \pi_{i\hat{\jmath}}^t (1 - \pi_{i\hat{\jmath}}^t) \mathbf{v}_{\hat{\jmath}}^t \mathbf{v}_{\hat{\jmath}}^{t^T} \ . \tag{6.2}$$

using moment matching [21]. Similarly, we approximate the posterior distribution of the latent feature of items $q(\mathbf{v}_j) = \mathcal{N}(\mathbf{v}_j|\hat{\mathbf{v}}_j, S_{\mathbf{v}_j}, \cdot)$. In practice, to efficiently compute the inverse of $S_{\mathbf{u}_i}, S_{\mathbf{v}_j}$, we used the Woodburry matrix identity [176].

Concretely, `Bernoulli Low Rank TS` proceeds as follows: At every round that user i interacts with the recommender system, we sample $\tilde{\mathbf{u}}_i$ from the posterior $\mathcal{N}(\mathbf{u}_i|\hat{\mathbf{u}}_i, S_{\mathbf{u}_i}, \cdot)$, and for every arm $j \in \mathcal{L}_i$, we sample $\tilde{\mathbf{v}}_j$ from the posterior $\mathcal{N}(\mathbf{v}_j|\hat{\mathbf{v}}_j, S_{\mathbf{v}_j}, \cdot)$. We show the item $\hat{\jmath} = \arg\max_j \tilde{\mathbf{u}}_i^T \tilde{\mathbf{v}}_j$ (as sigmoid is a monotonic function). The user provides feedback $r_{i\hat{\jmath}}$. Based on this feedback, we update $\hat{\mathbf{u}}_i$, $S_{\mathbf{u}_i}$, $\hat{\mathbf{v}}_{\hat{\jmath}}$, $S_{\mathbf{v}_{\hat{\jmath}}}$, thus updating the posteriors. At the next round, the posteriors become the new priors.

6.4.2 Collaborative Cascade Ranking Bandits

Here, we propose a novel algorithm assuming the low rank CF principle for the cascade list recommendation setting. Recall that the cascade list setting is suitable for systems learning to interact with users in a top-K list recommendation setup (Section 6.2.5). We model the reward of an item j for user i with $\mathbb{E}[r_{ij}] = \mathbf{u}_i^T \mathbf{v}_j$ for Gaussian rewards, and $\mathbb{E}[r_{ij}] = \mathrm{sigmoid}(\mathbf{u}_i^T \mathbf{v}_j)$ for Bernoulli rewards similarly to Section 6.4.1.

`Gauss/Bernoulli Low Rank Cascade TS` proceeds: At every user i-system interaction round we sample $\tilde{\mathbf{u}}_i \sim \mathcal{N}(\mathbf{u}_i|\hat{\mathbf{u}}_i, S_{\mathbf{u}_i}, \cdot)$, and $\tilde{\mathbf{v}}_j \sim \mathcal{N}(\mathbf{v}_j|\hat{\mathbf{v}}_j, S_{\mathbf{v}_j}, \cdot)$ $\forall$ candidate item

j. Sorting the estimated rewards based on the sampled values $\tilde{r}_{ij} = \tilde{\mathbf{u}}_i^T \tilde{\mathbf{v}}_j$, we present the list of items with the top K rewards. We observe the user's click on an item from the top-K list (if any), and update $\hat{\mathbf{u}}_i, S_{\mathbf{u}_i}, \hat{\mathbf{v}}_{\hat{j}}, S_{\mathbf{v}_{\hat{j}}}$ *for every item* $\hat{j}$ *at position* $\leq$ *the position of the click* using as feedback $r_{i\hat{j}} = 0$ for the skipped items, and $r_{i\hat{j}} = 1$ for the clicked item.

This new collaborative cascade ranking formulation does not need contextual features, and can learn the underlying implicit structure among users and items on-the-fly. This is what allows for effective propagation of the feedback across items and users via the learned embedding; and can lead to improved regret performance.

6.5 Experimental Results

We present our experiments with the goal of addressing the two following questions:

1. For one-item recommendation, which TS-based interactive recommendation algorithm is the most effective?
2. For top-K cascade list recommendation, how collaborative cascade ranking bandits compare to the other learning to interact algorithms?

6.5.1 Experimental Procedure

For each question we consider both Gaussian rewards and Bernoulli rewards, using the corresponding set of algorithms.[3]

Methods. We compare ten algorithms for the click/no-click setup—five for Gaussian and five for Bernoulli; and eight algorithms for the cascade click setup—four for Gaussian and four for Bernoulli rewards. Additionally, we included the baseline strategies of: (i) `Random`, an explore-only strategy, which at every round, selects for the incoming user i a random item from the set of active arms $\mathcal{L}_i$, and (ii) `Low Rank (LR) Greedy`, an exploit-only low rank CF strategy, which at every round, instead of sampling from the distributions $P(\mathbf{u}_i|\hat{\mathbf{u}}_i, S_{\mathbf{u}_i}, \cdot)$, $P(\mathbf{v}_{\hat{j}}|\hat{\mathbf{v}}_{\hat{j}}, S_{\mathbf{v}_{\hat{j}}}, \cdot)$, operates according to the distribution means $\hat{\mathbf{u}}_i, \hat{\mathbf{v}}_{\hat{j}}$, following precisely [114].

Evaluation Metric. We evaluate the algorithms in terms of *cumulative regret*, as

[3] A comparison among the two sets though is not applicable, due to the separate reward generation procedures.

Dataset	# Users	# Items	# Ratings
MovieLens 100K	943	1,682	100,000
MovieLens 1M	6,040	3,706	1,000,209
Netflix	5,000	500	1,922,815
Yahoo!	6,000	1,000	3,522,232

Table 6.1: Real Data Statistics.

learning to interact algorithms aim to minimize the difference between their actual performance and the optimal performance [152]. Formally, the cumulative regret over T rounds (Section 2.4) is

$$\mathcal{R}_T = \mathbb{E}\left[\sum_{t=1}^{T} r_{\mathbf{A}^*,\theta^*} - r_{\mathbf{A}_t,\theta_t)}\right], \tag{6.3}$$

where $\boldsymbol{A}^*$ is the *optimal* item/list maximizing the reward at any time t, $\boldsymbol{A}_t$ what the system actually showed at round t, and r denotes the reward. Smaller values of $\mathcal{R}_T$ are better, with 0 indicating optimal system performance.

Datasets. We used real recommendation data, whose statistics are shown in Table 6.1. Particularly, we considered three movie datasets: MovieLens 100K, MovieLens 1M, Netflix and a musical artist dataset from Yahoo! (Section 2.6).

Offline to Interactive Setup. Evaluating learning to interact algorithms on offline datasets, is a known issue and an active research topic [105]. Most public recommendation data, including the ones considered in Table 6.1, contain ratings on user-item interactions without giving access to the counter-factual response, i.e., how every user would have interacted with other items, had they been shown to them. Although rejection sampling and importance sampling techniques have been proposed, they typically require large-scale exploration data [104, 105]; however, the collected recommendation data are far from being purely exploratory. Thus, adopting the setup used in other interactive recommendation works (e.g., [91]), we incrementally showed entries of the observed rating matrix, as follows:

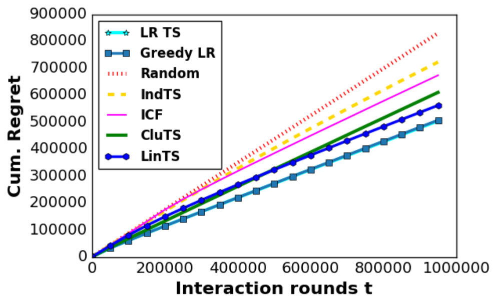
At the 0-th interaction round, no entry of the reward matrix R has been revealed. At every round t, we randomly sample one of the users present in the data i_t to interact with the system; this can be a user the system has already interacted with in previous rounds (warm-start), or a new one (cold-start). We consider as the active set of arms for user i, i.e., $\mathcal{L}_i$, only the items he has rated in the original dataset. The under evaluation algorithm decides which arm(s)/item(s) to recommend to the user.

Feedback Simulation. We simulate the user's feedback on the shown item $\hat{j}^t$ using the two reward models discussed in Section 6.2.3. For this, we need access to (i) the true underlying parameters $\boldsymbol{\theta}^*$ and (ii) for Gaussian rewards, the noise level. For (i), we performed a transformation of the rating values of the original datasets. The original MovieLens 100K and 1M contain ratings in a scale of 1 to 5 stars, while Netflix and Yahoo! have ratings in a 1 to 6 scale. To set the μ_{ij}^* of Gaussian rewards, we converted all ratings ≥ 4 to 1 (user likes the item), and < 4 to 0.01 (user dislikes the item). For Bernoulli rewards, we set the true user probability of liking an item π_{ij}^* by transforming the original ratings of scale $[1, 5 \text{ or } 6]$ to the scale $[0, 1)$ so as to represent probabilities. For this, we used the mapping $[4.5, 6] \rightarrow 0.9$, $[4, 4.5) \rightarrow 0.85$, $[3.5, 4) \rightarrow 0.8$, $[0.5, 3.5) \rightarrow 0.05$, $[0, 0.5) \rightarrow 0$ which indicates that the higher the rating, the more likely the user will click on the item. For (ii), for the one/no-click model, we set noise σ_{ij} to 0.5 $\forall i, j$; in contrast, for the cascade click model we set noise to 0 (1 is a click, and 0.01 a skip).

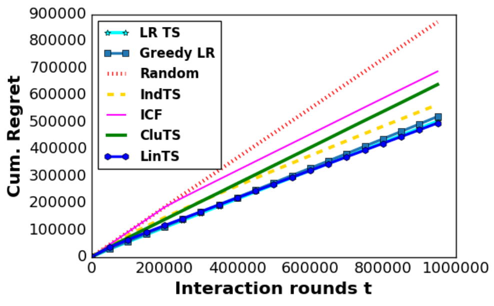
Parameter Setting. We set the number of interaction rounds T to 1,000,000. We initialize the parameters of the prior distributions as follows: For `Independent (Cascade) TS`, we set the prior to a broad one, i.e., $\mathcal{N}(0, 1)$. For `Bernoulli Low Rank (Cascade) TS`, we set the variance of the user and item latent features to 0.1, i.e., $\sigma_u^2 = \sigma_v^2 = 0.1$. For `Gauss Low Rank (Cascade) TS`, we set σ_u^2, σ_v^2 to 0.001. These choices were made as we found that larger variance values can lead to deteriorated performance. In any method assuming low rank structure, a rank of 2 is used, similarly to [91]. For clustered bandits, the number of clusters is set to 10. In practice, one should do parameter tuning. In the Appendix A.2, we explore `Low Rank TS`'s sensitivity to the various parameters.

6.5.2 Which interactive recommender is the best for one/no-click recommendation?

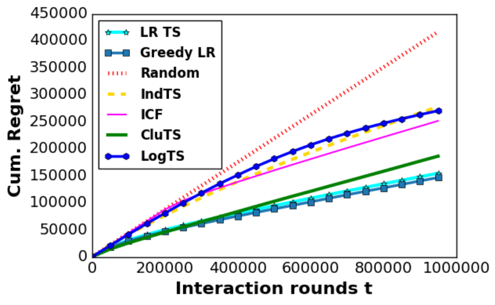
Figure 6.1 shows the performance of the learning to interact recommendation algorithms for the one/no-click setup. Results only for MovieLens 1M and 100K (Gauss and Bernoulli) are shown for brevity. In the Appendix A.3 we include results for all datasets for completeness.



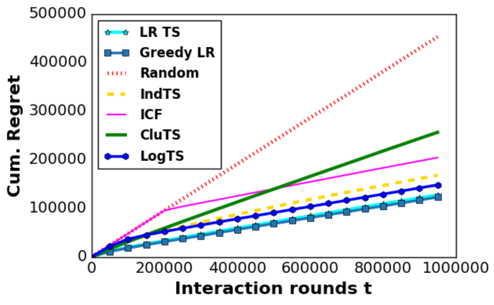
(a) MovieLens 1M, Gauss



(b) MovieLens 100K, Gauss



(c) MovieLens 1M, Bernoulli



(d) MovieLens 100K, Bernoulli

Figure 6.1: Comparison of learning to interact algorithms for the *One Item* setup w.r.t cumulative regret. The low rank hypothesis achieves the best performance, except for MovieLens 100K Gauss, where `LinTS` is the best.

Low rank vs. Independent.

In all datasets, both for Bernoulli and Gaussian rewards, `Low Rank TS` is (among) the best strategies. Particularly, for datasets with a larger number of observations (MovieLens 1M, Netflix and Yahoo!), `Low Rank TS` seems to couple users and items in the low dimensional embedding effectively; thus making the value of the low rank representation more prominent. In these cases, `Low Rank TS` clearly outperforms `Independent TS` (`IndTS`), whose performance is close to `Random`. In contrast, for datasets with fewer observations (MovieLens 100K), the gap between `Low Rank TS` and `Independent TS` closes.

Greedy vs. TS.

For Gaussian rewards, for MovieLens 100K and MovieLens 1M `LR Greedy` performs closely to `Low Rank TS`—this is due to the small value of noise; while for larger scale datasets `Low Rank TS` is the best. For Bernoulli rewards, `LR Greedy` surpasses `Low Rank TS`, indicating that the low rank part is more important than the exploration part. This happens because we set π^*_{ij} very close to the extremes of 0 and 1.

Latent context vs. Explicit context.

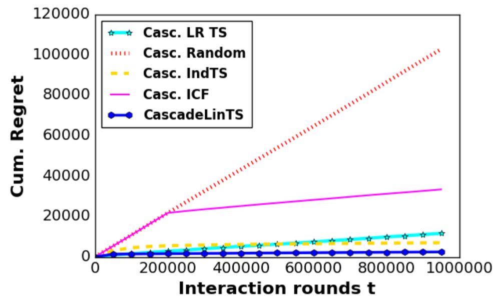
Recall that `CluLinTS`, `LinTS`, `CluLogTS` and `LogTS` are the only algorithms which use the contextual information. We evaluate these algorithms' performance only for MovieLens 100K and MovieLens 1M, as these are the only datasets which contain explicit context. We used as contextual features the items' genre features, since it has been found that genre correlates well with the users' ratings for these datasets [156]. Every such feature is represented as an one-hot encoding of the genres characterizing the movie, and has dimension the total number of genres (19 for MovieLens 100K and 18 for MovieLens 1M).

We can see that for MovieLens 100K, the genre feature is informative enough, as after about 400,000 interaction rounds `LinTS` outperforms `Low Rank TS`. However, for the larger dataset of MovieLens 1M, even though `Low Rank TS` does not take advantage of the explicit context information, it is able to surpass `LinTS` by learning both user and item features. For Bernoulli rewards, for both datasets, treating recommendation as a low rank bandit problem instead of a contextual linear bandit is better.

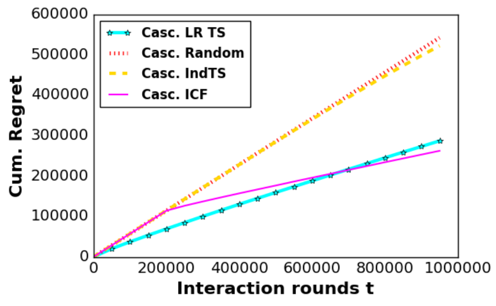
Comparing the contextual algorithms, we see that for Bernoulli rewards, sharing feedback among the clusters outperforms having independent `LogTS` per user for MovieLens 1M, whereas the opposite trend is found in MovieLens 100K. For Gaussian rewards, in both datasets `LinTS` is better than `CluLinTS`, but the performance gap closes for the larger MovieLens 1M.

Fixing V vs. learning V.

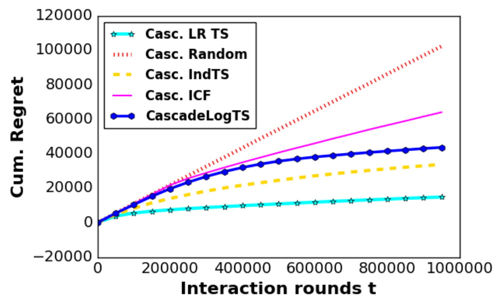
Recall that `ICF` uses an independent `LinTS` per user using as contextual features prelearned item latent features $\boldsymbol{V}$. Following [91], for `ICF` we learn $\boldsymbol{V}$ by performing online Gaussian/Logistic matrix factorization in the first $20\%T$ of the interaction rounds, during which we randomly show an item while updating $\mathbf{v}_j$ and the user covariance matrix



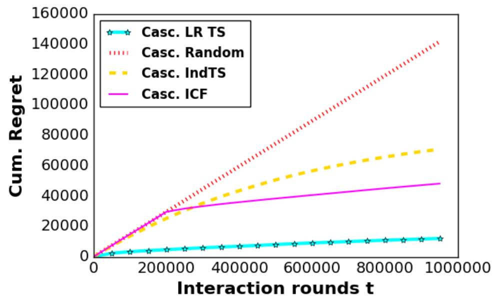
(a) MovieLens 100K, Gauss



(b) Yahoo!, Gauss



(c) MovieLens 1M, Bernoulli



(d) Netflix, Bernoulli

Figure 6.2: Comparison of learning to interact algorithms for the *Cascade List* setup in terms of cumulative regret. For larger datasets, the low rank principle for the cascade ranking setup achieves the smallest regret.

$S_{\mathbf{u}_i}$. After 20%T rounds, we fix $\mathbf{v}_j$ and we show the item $\hat{j} = \arg\max_j \tilde{\mathbf{u}}_i^T \mathbf{v}_j$. In all the datasets, learning the latent features of both users and items via `Low Rank TS` surpasses `ICF` both for Gaussian and Bernoulli rewards.

6.5.3 Which interactive recommender is the best for cascade list recommendation?

Figure 6.2 shows how collaborative cascade bandits compare to the rest cascade learning to interact algorithms. Based on results from all datasets (although here we show only the performance for MovieLens 1M and Netflix for Bernoulli rewards, and for MovieLens 100K and Yahoo! for Gaussian rewards), we observe the following:

Independent vs. Low Rank.

When enough observations are available (MovieLens 1M, Netflix, Yahoo!), `Low Rank Cascade TS` outperforms `Independent Cascade TS`. But when the number of these observations is small (MovieLens 100K), `Independent Cascade TS` is better.

Latent context vs. Explicit context.

For MovieLens 100K, using the genre feature as context is enough to achieve quite good performance; particularly, for Gaussian rewards, `CascadeLinTS` is the best performing strategy. In contrast, in the larger dataset of MovieLens 1M, learning on-the-fly the latent attributes of users and items via `Low Rank Cascade TS` results in a smaller regret compared to `CascadeLinTS`/`CascadeLogTS` for either reward setting.

We conclude that for both Bernoulli and Gaussian rewards, `Low Rank Cascade TS` is the best performing strategy for large-scale datasets. The only exception is the Yahoo! dataset for Gaussian rewards, where `ICF Cascade TS` surpasses `Low Rank Cascade TS`; which however again is a *low rank* cascade bandit strategy.

6.6 Related Work

Here we highlight some pairwise comparisons among the various learning to interact for recommendation algorithms which have happened in the literature: (1) Li et al. devised contextual linear bandits and showed that their proposed algorithm `LinUCB` outperforms several baselines (e.g., popular, random, user segment-based algorithms) [104]. (2) The authors of [125] using the clustering parametric assumptions on top of `LinUCB`, devised `CluLinUCB` and showed that it outperforms having an independent `LinUCB` for each user, and performs similarly or better than the closely related baseline proposed in [69]. (3) Zhao et al. proposed `ICF`, posing recommendation as a contextual linear bandit with context the learned item features, and showed that it outperforms interview-based and active learning strategies with a few user-system interactions [184]. (4) In [91], Kawale et al. showed that combining TS with online matrix factorization, while maintaining particle filters to sample from the posterior, outperforms `ICF` as well as alternative ways of posterior sampling. A similar model was proposed for conversational recommender systems [51]. (5) In [106], Li et al.'s collaborative filtering bandits, which implement the CF view via co-clustering, were shown to surpass `LinUCB` and the clustering bandits in [36, 69]. Other relevant works are [93, 171, 177, 68], which all combine the clustering or CF principle with contextual features on the Gaussian rewards, one-item setup.

Other works balancing the EE trade-off in recommendation are: [6] which gives bayesian schemes for serving content, [169] which combines Gaussian processes with EE, [30] which proposes an ϵ-greedy user-user neighbor-based CF method, [154] which gives an ϵ-greedy online matrix factorization method, and [173] which combines CF with TS and topic models.

6.7 Summary

In this chapter, we tackled the problem of learning to interact with users, which is central in recommender systems, and made the following contributions:

- **Algorithms:** We (a) developed interactive learning recommendation algorithms suitable for *dyadic data*, and (b) offered the novel *collaborative cascade bandits.*
- **Experiments:** (a) We provided an extensive study making all pairwise comparisons among various learning to interact algorithms for recommendation: 10 algorithms for one-item (5 for Bernoulli, 5 for Gaussian), and 8 algorithms for cascade list (4 for Bernoulli, 4 for Gaussian). (b) We showed that *for large-scale data, collaboratively learning to interact algorithms are the best performing ones.*

Part III

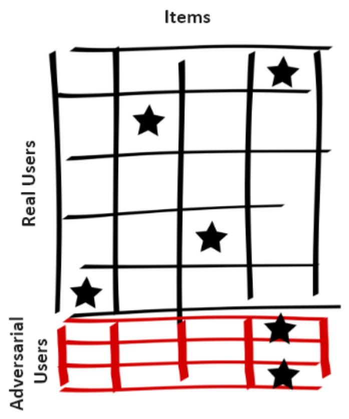
(Malicious) Data: The Attack of The Fake Users

Based on work in [47].

Part III: Overview

So far in this dissertation, similarly to most works on recommendation, we have assumed that the users in the recommendation system give their data, either in the offline setting (Part I), or in the interactive setting (Part II), with good intentions—truthfully reporting what they like or they do not like, or browsing according to their actual preferences.

However, what happens if there is a small proportion of users in the recommendation system who have some *adversarial goal*, e.g. target a certain subset of users or items? How does this affect the recommendation system's performance, especially when the model parameters are learned collaboratively, relying on the idea that similar users tend to like items similarly?



Sketch of an adversarial attack in a recommender system.

Such a scenario is called an adversarial attack in recommendation systems, relying on including a small set of fake user profiles, hand-engineered to target a specific item. Typically, this *hand-coding* consists of having the fake users use the maximum allowed score for the item they target, and specify random or normal-distributed scores in a subset of the items $\mathcal{I}$ [127, 98].

Recently, with the great successes of neural networks in image recognition, and the wide door of possibilities of applying such tools in everyday life, a line of work has emerged on how to use machine learning to learn adversarial attacks against neural networks so to mis-classify images with high probability [164, 74, 122]. Learning such attacks is step one towards safer and more trustworthy machine learning systems; step two is ensuring that the systems are robust to such learned attacks.

It is only natural then, to ask whether in the context of recommendation systems we can perform *machine learned adversarial attacks.* This is the focus of the following chapter.

Chapter 7

Recommendation as a Game with an Adversary

How vulnerable are recommendation models to ***machine learned*** *adversarial attacks?*

Can machine learning models for recommendation be easily fooled? While the question has been answered for hand-engineered fake user profiles, it has not been explored for *machine learned* adversarial attacks. This chapter attempts to close this gap.

We propose a framework for generating fake user profiles which, when incorporated in the training of a recommendation system, can achieve an *adversarial intent*, while remaining *indistinguishable* from real user profiles. We formulate this procedure as a repeated general-sum game between two players: an oblivious recommendation system R and an adversarial fake user generator A with two goals: (G1) the rating distribution of the fake users needs to be close to the real users, and (G2) some objective f_A encoding the attack intent, such as targeting the top-K recommendation quality of R for a subset of users, needs to be optimized. We propose a learning framework to achieve both goals, and offer extensive experiments considering multiple types of attacks highlighting the vulnerability of recommendation systems.

7.1 Introduction

Fake social media accounts are created to promote news articles about a political ideology; false online product reviews attempt to bias users' opinions favorably or against certain products—these are just a few of the many real life examples illustrating that recommendation systems are exposed and can be susceptible to threats from adversarial parties.

Machine learning algorithms have an ever-growing impact on people's everyday lives. Recommendation systems heavily rely on such algorithms to help users make their decisions—from which show to watch, to which news articles to read (which could end up influencing their beliefs). Thus, a natural question is: *How easy is it to manipulate a machine learned system for malicious purposes?* An answer to such a question would be a stepping stone towards safer artificial intelligence [130, 71].

To study this question, the first necessary step is the creation of adversarial examples; this would allow to test the algorithms against them, and potentially increase the algorithms' resistance [74]. With this motivation, a recently thriving subfield of machine learning is the one of adversarial examples—find the minimal perturbation vector to add to the feature vector of an example so that an oblivious classifier misclassifies the perturbed example. These works focus on classification [164, 74, 122, 121, 161].

In *recommendation systems*, the adversarial attacks have a different form. Instead of minimally perturbing an existing example to misclassify it, the attack consists of creating a few adversarial user profiles rating items with some intent. The intent could be to promote a specific item, or to deteriorate the recommendation quality of a group of users. The setting is not new; in fact, it has been researched since [127, 98]. However, the injected fake user profiles are hand-coded—typically, the fake users rate the target item with a small or large score, and the rest with random or normal distributed scores to mimic the true rating distribution.

Our goal is to revisit the question of crafting adversarial fake user profiles for a recommendation system *from an optimization perspective*. We pose this as finding a matrix of fake users$\times$items, so that (G1) the distance between the distributions of real and fake users is small, and (G2) the adversary's intent is accomplished, e.g. removing a group of news articles of the opposing ideology from users' top-K recommended lists. The scenario is highly realistic—e.g., an adversary A creates a small number of realistic-looking fake user accounts with the goal of removing a target group of a competitor company's products from target users' top lists. We assume that A knows the recommender's model and algorithm to fit the model, and can fit similar models on any new/fake data.

Particularly, we make the following contributions:

1. We formulate *adversarial recommendation* as a game of an adversary vs. an oblivious recommender, e.g. a low-rank model. There are two objectives: (1) given the real and some fake ratings, learn the low-rank model based on the *recommender's objective* and (2) use the low-rank model to evaluate the *adversarial objective*. This two-step process makes the adversary's task more involved.
2. We propose a **learning framework** for adversarial attacks on recommendation systems, using: (i) generative adversarial nets (GANs) [73] to learn initial fake users that mimic the true rating distribution and (ii) suitably update them optimizing an objective encoding the adversarial goal. For (ii), we use *0-th order optimization* to construct the gradient, as the adversary does not have direct access to the gradient.
3. This is the first time to pose finding fake user profiles in an optimization setting, and thus finding *machine learned* attacks on recommendation systems; this allows optimizing complex objectives in a unified way, even in the absence of gradient information.
4. Our **real-world experiments** show that machine learned adversarial attacks with a wide range of intents are very much possible. As a striking example of a malicious attack we illustrate that in order to ruin the predicted scores of a specific item for users who would have loved or hated that item, it suffices to minimize the predicted score of the user with the highest predicted score before the attack.

The rest of the chapter is organized as follows. In Section 7.2 we formalize the problem of adversarial recommendation and in Section 7.3 we propose our learning procedure from the perspective of an adversary of the recommender. We empirically evaluate the proposed methods in Section 7.4, review related work in Section 7.5, and give a summary in Section 7.6.

7.2 Problem Formulation

We begin with a discussion of the model for attacking a recommendation system. The model involves two players: an *oblivious* recommendation system R and an *adversarial* 'fake user' generator A. The goal of the recommendation system R is to build a model with parameters θ_R to minimize a suitable loss function between true and model predicted ratings over all users and items. The goal of the adversary A is to generate fake users using a model with parameters θ_A such that:

(G1) the fake users are indistinguishable from the real users based on reasonable metrics, e.g., ratings distributions of the fake users are similar to real users, eigenspectrum of the fake user ratings are similar to that of real user ratings, etc., and

(G2) a recommendation model learned by R using the fake users generated by A leads to worse predicted ratings for a suitable subset of the real users and/or items, e.g., makes an item less desirable to a subset of users.

Let $\mathcal{I}$ be the set of items, and $\mathcal{U}$ the set of real users present in the recommendation system. Let $m = |\mathcal{I}|$ be the number of items and $n = |\mathcal{U}|$ the number of real users, where $|\cdot|$ denotes the cardinality of a set. Let $X \in \mathbf{R}^{|\mathcal{U}| \times |\mathcal{I}|}$ denote the matrix of ratings from real users. We assume that the adversary A has a certain budget of k fake user profiles, where $k \ll |\mathcal{U}|$; and that each user profile is a $|\mathcal{I}|$ dimensional vector; aka how the user has rated the different items in $\mathcal{I}$, with zero values denoting empty ratings. Particularly, A outputs a matrix $Z \in \mathbf{R}^{k \times |\mathcal{I}|}$, where each row $\mathbf{z}_{i'}$ for $\{i'\}_1^k$ is a fake user profile. The total of real and fake users is $n' = n + k$.

The setting can be formulated as a repeated, general-sum game between two players: the *row* player, the recommender R and the *column* player, the adversary A. The recommender R maps (u, j, r) tuples to some real-valued score representing the predicted rating of user u on item j, and is parameterized by θ_R. The actions of R include all θ_R, e.g., for low rank recommender models, each θ_R corresponds to a pair of U, V latent factor matrices. The actions of the adversary A include all fake user profiles Z, which are generated using a model parameterized by θ_A.

Both players consider a loss function (the negative of a payoff function) they wish to minimize. If the row player chooses actions $\tilde{U}, \tilde{V}$ (latent factor matrices) and the column player chooses action $\tilde{Z}$ (fake user matrix), then for the row player, the functional form of the payoff is $f_R(\tilde{U}, \tilde{V}, \tilde{Z})$, and for the column player, is $f_A(\tilde{U}, \tilde{V}, \tilde{Z})$.

In the general setting, each player maintains a distribution over respective action spaces, and will play by drawing an action from the distribution. The row player maintains a distribution P_r over the space of (U, V), i.e., $(\tilde{U}, \tilde{V}) \sim P_r(U, V)$, and the column player maintains a distribution P_c over Z, i.e., $\tilde{Z} \sim P_c(Z)$. The distributions of the recommender and the adversary are parameterized by θ_R and θ_A respectively. In a repeated game setting, let (θ_R^t, θ_A^t) be the current parameterizations of the two players. In the next step, the goal of each player is to find optimal parameters θ_R^{t+1} and θ_A^{t+1}

respectively such that their corresponding expected loss is minimized:

$$\theta_R^{t+1} = \underset{\theta_R}{\mathrm{argmin}} f_R(\theta_R, \theta_A^t), \quad \theta_A^{t+1} = \underset{\theta_A}{\mathrm{argmin}} f_A(\theta_R^t, \theta_A)$$

or

$$\begin{aligned} \theta_R^{(t+1)} &= \mathbf{E}_{(U,V)\sim P_r(\theta_R), Z\sim P_c(\theta_A^t)} \left[f_R(U,V,Z) \right] \\ \theta_A^{(t+1)} &= \mathbf{E}_{(U,V)\sim P_r(\theta_R^t), Z\sim P_c(\theta_A)} \left[f_A(U,V,Z) \right] \ . \end{aligned}$$

Note that $f_R \neq -f_A$, so the game is not zero sum.

However, one main challenge for A is that there is a two-step process going on: (step 1) given some fake ratings, learning say the low-rank model based on the recommendation system objective, typically using non-convex optimization, and (step 2) use the low-rank model to evaluate the adversarial objective. As a result, the adversary typically cannot compute the gradient of the effect w.r.t. Z. In the sequel we approach the problem of constructing Z from the adversary's perspective.

7.3 Learning

We detail the specifics of the recommender and adversary considered, and discuss our proposed learning approach.

Recommender Strategy. We assume throughout that the recommender R is *oblivious* to the existence of an adversary, hence, it optimizes its loss over *all* given data—before the attack over only the C_{real} original training user-item-rating tuples $\{u_c, j_c, y_c\}_{c=1}^{C_{\text{real}}}$; after the attack over both $\{u_c, j_c, y_c\}_{c=1}^{C_{\text{real}}}$ and the C_{fake} non-zero ratings of the k fake user profiles, succinctly represented as a sparse matrix $Z \in \mathbf{R}^{k\times m}$ produced by the adversary$A$, resulting in an augmented training set of $\{u_c, j_c, y_c\}_{c=1}^{C_{\text{all}}}$, with $C_{\text{all}} = C_{\text{real}} + C_{\text{fake}}$. In particular, R, using parameters θ_R and a goodness-of-fit loss function $\ell(\cdot)$, maps input tuples $\{u_c, j_c, y_c\}_{c=1}^{C^{\text{all}}}$ to estimated scores $\{\hat{y}_c\}_{c=1}^{C_{\text{all}}}$, so that the loss f_R is minimized

$$\min_{\theta_R} f_R(\theta_R, \theta_A) = \min_{\theta_R} \frac{1}{C^{\text{all}}} \sum_{c=1}^{C_{\text{all}}} \ell(y_c, \hat{y}_c(u_c, j_c; \theta_R, \theta_A)).$$

We assume that the recommender is a low rank model; however, our overall approach is not specific to such a model. The low rank recommender has latent factors $U \in \mathbf{R}^{n'\times d}$

capturing the latent preferences of users, and $V \in \mathbf{R}^{m \times d}$ capturing the latent attributes of the items, and optimizes its expected loss over its parameters U, V:

$$(U^*, V^*) = \arg\min_{U,V} \|[X; Z] - UV^T\|_2^2 + \lambda(\|U\|_2^2 + \|V\|_2^2) \tag{7.1}$$

where $[;]$ denotes concatenation of two matrices over the row axis. We optimize the loss by alternative minimization (alt-min for short), i.e., alternating the closed-form U, V update equations, for a few iterations [119]. The model has a probabilistic interpretation: the prior parameter distributions are $\mathbf{u}_i \sim \mathcal{N}(0, \lambda I), \mathbf{v}_j \sim \mathcal{N}(0, \lambda I)$ and conditional model $X(i, j) \sim \mathcal{N}(\mathbf{u}_i^T \mathbf{v}_j, \sigma)$, where I is the $d \times d$ identity matrix, $\mathbf{u}_i$ is the i-th row of U, and $\mathbf{v}_j$ the j-th row of V. Thus, from a Bayesian perspective, R can maintain a posterior distribution over its parameters (U, V). For computational simplicity, R is assumed to pick the mode (U^*, V^*) of the posterior distribution.

The adversary A is aware of the model and algorithm, and is able to compute the point estimates (U^*, V^*) for any chosen Z. Note that since an alt-min algorithm is needed to obtain (U^*, V^*), the adversary does not have a direct way to do gradient descent w.r.t. Z on functions of (U^*, V^*); we will return to this point later.

Adversary Strategy. The adversary A, with parameters θ_A, learns and outputs a (distribution over) fake user matrix $Z \in \mathbf{R}^{k \times m}$, which should satisfy the two goals presented earlier—(G1) the unnoticeability goal and (G2) satisfying an adversarial intent. The intent is captured by the loss function $f_A(\theta_R^t, \theta_A)$; various intents capturing different attack strategies are explored in our experiments. The intent can be defined over a set of target items $\mathcal{I}_{\mathrm{H}}$, target users $\mathcal{U}_{\mathrm{H}}$, a single target user u, or target item h. Some examples of intents are:

- Target the predicted score for $(u,\, h)$: $f_A = \hat{y}(u, h)$
- Target the mean predicted score for h over the target users $\mathcal{U}_{\mathrm{H}}$:
  $f_A = \frac{1}{|\mathcal{U}_{\mathrm{H}}|} \sum_{u \in \mathcal{U}_{\mathrm{H}}} \hat{y}(u, h)$
- Target recommendation metric@top for $\mathcal{U}_{\mathrm{H}}$:
  e.g. Hit Rate (HR), $f_A = \frac{1}{|\mathcal{U}_{\mathrm{H}}|} \sum_{u \in \mathcal{U}_{\mathrm{H}}} \mathrm{HR}(u)$.

Approach for (G1). We generate fake users by using generative adversarial nets (GANs) [73]. In GANs, a pair of Generator-Discriminator networks pitty each other—the generator G generating samples with the goal of fooling the discriminator D to not being able to distinguish them from real. At convergence, the conditional distribution of the generator G should give fake user samples which D cannot discriminate from real ones. We discuss details of the GANs architecture in the experiments.

Approach for (G2). To accomplish both (G1) and (G2), one could in principle change the loss of GANs to be a convex combination of two losses: the "perception" loss of fooling D (as in GANs formulation) and the "adversarial loss" encoding the adversary's intent, i.e., f_A.

Instead, we opt for a simpler two-step approach: first train GANs until convergence and sample a set of fake users $Z_1 = Z_{\text{GAN}}$ from the conditional posterior of G; and second, suitably modify the sampled users using a variant of gradient descent to optimize the adversary's intent f_A over the fake users Z. In the process, we want to make sure that the resulting fake users helping with the adversary's intent do not come across as obviously fake.

Let us consider the problem of interest to the adversary:

$$\min_Z f_A(Z), \tag{7.2}$$

where recall that Z plays the role of the θ_A actions of the adversary and the argument θ_R^t, i.e., U^t, V^t, is dropped for brevity. To optimize (7.2) we use *projected gradient descent* for $\{t\}_1^T$

$$\tilde{Z}_{t+1} = Z_t - \eta \nabla_{Z_t} f_A(Z; \theta_A),\ Z_{t+1} = \Pi_{\text{allowed range}}(\tilde{Z}_{t+1}) \tag{7.3}$$

where the projection Π is to ensure that the marginals of real and fake users remain close after the descent, η is the learning rate, and $\nabla_{Z_t} f_A$ is the gradient of the adversarial loss w.r.t Z_t.

Now the question is, how can we compute the gradient $\nabla_{Z_t} f_A$? To make things concrete, let us consider as adversarial intent: minimize the predicted rating of item h over all real users who have not rated the item

$$\min_Z \frac{1}{|\{u \notin \text{Ra}(h)\}|} \sum_{u \notin \text{Ra}(h)} \mathbf{u}_u^T \mathbf{v}_h \tag{7.4}$$

At first glance, from (7.4) the loss is a function of the recommender's parameters, and not Z. But, recall from (7.1) that the parameters of the recommender *are* a function of Z—more generally, f_A is a function of (θ_A, θ_R^t). After playing fake matrix Z, the adversary player gets to observe the loss *only for this single Z played*, and not the other actions/ matrices it could have played. Thus, the adversary gets limited information, or else *bandit feedback*. Put differently: the gradient of the loss is not directly given for the optimization over Z.

To obtain an *approximation of the gradient*, we build upon 0th-order optimization works in bandit optimization [3, 62]. The idea is that if we can only perform query evaluations, to obtain the gradient of $f(Z)$, we need to query $f(Z)$ at two nearby points: Z_t and $Z_t + \alpha Z_0$, for a small α and a suitable fixed matrix Z_0. Then we can compute the gradient as the directional derivative along the direction Z_0: $\nabla f(Z_t) = (f(Z_t + \alpha Z_0) - f(Z_t))Z_0/\alpha$.

We use a refinement of Algorithm 3 in [3]: there instead of two-point evaluation, they needed K directions and computed the gradient using all K directions. Instead we use as K directions the K top left and right singular vectors of the fake user matrix at round t Z_t, obtained from a Singular Value Decomposition on Z_t: $Z_t = \tilde{U}\Sigma\tilde{V}^T$. Let $Z^{(h)}$ be the rank one matrices built from each left and right singular vectors of Z_t, $Z^{(h)} = \tilde{u}_h\tilde{v}_h^T$ for $\{h\}_1^K$, where K is the rank of Z_t. Then, we can use these rank-1 matrices $Z^{(h)}$ as K possible directions, and compute the matrix gradient based on these:

$$\nabla f(Z_t) = 1/\alpha \sum_{h=1}^{K} (f(Z_t + \alpha Z^{(h)}) - f(Z_t))Z^{(h)} \tag{7.5}$$

This involves $K + 1$ evaluations of the function $f(Z)$. To make things faster, we use warm-start—we first evaluate $f(Z_t)$, then, for any $f(Z_t + \alpha Z^{(h)})$, we use the final (U, V) for $f(Z_t)$ to warm start the iterates. Similar strategies have been studied in stochastic and evolutionary optimization [26, 78].

Algorithm 5 summarizes our proposed learning approach.

Algorithm 5 Learning Algorithm for Adversary's Strategy

```
1: $U^0, V^0 \leftarrow$ train low rank $R$ over real data $X$.
2: $Z^0 \leftarrow$ train generative adversarial nets on the dataset.
3: for  $t = 1, \dots, T$ do
4:     Update the adversary's $Z$ descending its gradient (7.5).
5:     for each of the $K + 1$ evaluations $f_A(Z)$ do
6:         Update the recommender's $(U, V)$ with alt-min (7.1).
7:     end for
8: end for
```

Dataset	# of Items	2D $H \times W$ Shape	# of Users
MovieLens 100K	1682	29×58	943
MovieLens 1M	3706	34×109	6040

Table 7.1: Dataset Statistics. The 2D Shape is the conversion of the $|\mathcal{I}|$-d vector to a 2-D H $\times$ W "image".

7.4 Experiments

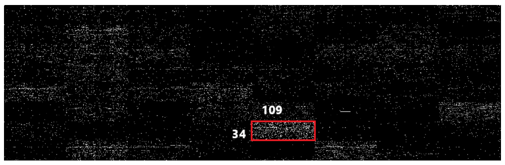
We design our experiments to understand the effectiveness of the proposed approach in creating an adversary model A that produces fake users which cannot be distinguished from real, and which influence in some way the recommender R.

7.4.1 Can We Learn Realistic User Profiles?

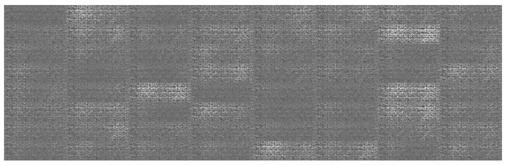
The first question we investigate is whether with generative adversarial nets we can learn fake user profiles that seem like real.

Network Architecture. We used the popular DCGAN architecture [134]. The Discriminator D takes an image of size $H \times W$ (fake or real user sample) and outputs either a 0 or a 1 (is it fake or real?). It consists of four 2D convolutional `CONV` units, with `leaky ReLUs` and batch normalization (`BN`), whose depths are respectively $[64, 128, 256, 512]$, followed by a single-output fully connected (`FC`) unit with sigmoid activation. The Generator G takes as input noise $z \sim \mathcal{N}(0, 100)$ and outputs an $H \times W$ image (the fake user sample). It consists of a `FC` unit of dimension 2×4 (or 7) $\times 512$ with `ReLU` and `BN`, reshaped to a $2, 4$ or $7, 512$ image, followed by four `transposed CONV` units of depths $[256, 128, 64, 1]$ respectively, each with `ReLU` and `BN`, except for the final with a tanh. We set for the (transposed) `CONV` units the stride to 2, the kernel size to 5×5. We set batch size to 64, and run DCGAN for 100 epochs (each epoch does a cyclic pass).

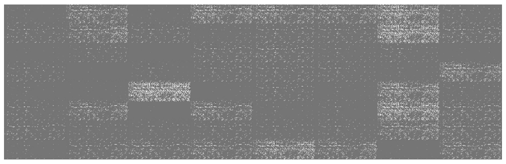
Datasets. Since we want to learn user profiles for recommendation systems, we used two popular movie recommendation datasets (Section 2.6), MovieLens 100K, MovieLens 1M [79], whose statistics are shown in Table 7.1. They contain the ratings of users on different movies in the scale $\{0, 1, 2, 3, 4, 5\}$ with 5 the highest like, and 0 denoting that the user has not rated the movie. As the last layer of G has the tanh activation function, fake user samples are in $[-1, 1]$; hence, using $r' = (r - 2.5)/2.5$ we transformed the real



(a) Real User Sample (first 64 users)



(b) Epoch 0



(c) Epoch 20

(d) Epoch 90

Figure 7.1: Visualization of 64 sampled fake users for MovieLens 1M in a 8×8 grid. The red box in (a) surrounds one real user sample, which is of dimension 34×109—when flattened it is 3706-dimensional, i.e., $|\mathcal{I}|$. As the training epochs progress, the generator network G of GANs learns to generate realistic-looking fake users.

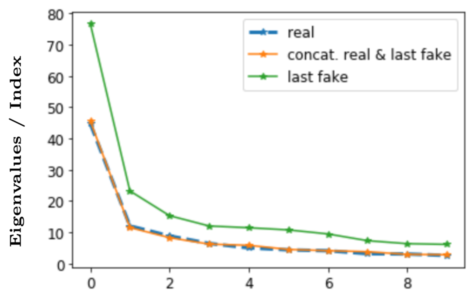
ratings from $[0, 5]$ to $[-1, 1]$.

Setup. Each user profile is an $|\mathcal{I}|$-d sparse vector, which needs to be transformed to a $H \times W$ 2D array to go through the DCGAN 2D (de-)convolutional units. We set as H the smallest factor of $|\mathcal{I}|$ and as $W = |\mathcal{I}|/H$. This way, each user is viewed as a 2D $H \times W$ image with pixel values the ratings of the user on the different items.

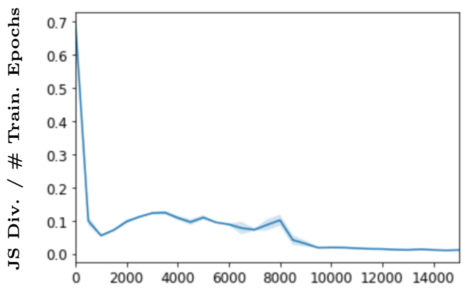
Results. Periodically during training, we sample 64 fake users from the conditional posterior of the generator G and visualize them in a grid of 8 by 8 $H \times W$ images. In Figure 7.1 we illustrate the progress of the samples during training for MovieLens 1M, with the first column visualizing the first 64 real users. We see that in the first epochs the sampled users are noise, but as training goes on, the real distribution seems to be learned; similar results hold for MovieLens 100K.

We also want to validate quantitatively that the fake user distribution is close to the real one at DCGAN's training convergence. For these experiments, we sample 700 fake users from G so that the size of the real and fake user distribution—at least for MovieLens 100K—is comparable.

We compute the correlation matrix over items of the fake data $Z^T Z$ (based on Z sampled from the conditional posterior of the learned G of the last training epoch),



(a) Top-10 Eigenvalues



(b) Jensen-Shannon (JS) Divergence

Figure 7.2: (a) x-axis: index of top eigenvalue, ranging from 0 to 9, y-axis: corresponding eigenvalue. The eigenspectrums of the correlation matrices of the real, the fake, and the [real; fake] concatenated data are similar. (b) x-axis: DCGAN training iteration, y-axis: mean JS. At convergence, the distance between the real and fake distributions is approximately 0.

and the correlation matrix of the real data X^TX, and we compare their respective eigenspectrum; specifically their **top-10 eigenvalues**.

We compute **distance metrics** between the real and fake user distributions: For each item j, the real users form a distribution P^j over the rating values $[-1.0, 1.0]$ (i.e., for MovieLens 100K with 943 samples), and the fake users $G(z)$ form a Q^j distribution over $[-1.0, 1.0]$ (with 700 samples). Then, we discretize the values to the six bins $[-1.0, -0.6, -0.2, 0.2, 0.6, 1.0]$ (corresponding to $[0, 1, 2, 3, 4, 5]$): for each of the six bins, we compute the fraction of (real or fake) users who have rated j in this bin out of all (real or fake) users. For a certain item j, we compute two such six-dimensional vectors, one for the real users, and a second for the fake users, and then use the following metrics to measure distance between distributions [70]:

$$\textbf{Total Variation Distance (TVD)}(P^j, Q^j) = \sum |P^j - Q^j|/2 \tag{7.6}$$

$$\textbf{Jensen-Shannon Divergence (JS)}(P^j, Q^j) = \frac{1}{2}(D(P^j||M^j) + D(Q^j||M^j)) \tag{7.7}$$

where $M^j = \frac{1}{2}(P^j + Q^j)$ and D is the Kullback-Leibler divergence. After we have computed the distance metric for each item, we report the average over all items, i.e, mean TVD and mean JS Div.

In Figure 7.2 we plot the (a) top-10 eigenvalues and (b) JS Divergence for the MovieLens 1M dataset. For (b), the reported results are averaged over five different runs of DCGAN. The results shown, along with similar results we have observed for MovieLens 100K, indicate our first key finding:

> *Generative adversarial nets can produce fake user samples whose distribution is close to the real user distribution.*

7.4.2 Experimental Design

To evaluate the adversary A's capability of attacking a recommender R, we use the two-phased approach described in Section 7.3: (1) We first train DCGAN on the respective dataset, as presented before in 7.4.1, and (2) we then perform the Z-SGD updates, which are initialized by a sample of DCGAN-generated fake users, denoted by Z_{GAN}, transformed from the $[-1, 1]$ to the expected by the recommender range of $[0, 5]$.

Throughout all experiments, the sample size of Z_{GAN} will be set to 64; the 64 fake users, iteratively optimized during the Z-SGD updates, represent only 0.063 fraction of all system users (real and fake) for MovieLens 100K, and 0.01 fraction for MovieLens 1M.

We use two types of experimental setups:

(E1) A targets unrated user-item entries (thus entries which are candidates for recommendation) not included in the training of R.

(E2) A targets a small subset from the recommender's true (user, item, rating) tuples, which is held out from the training of R.

For (E2), we define a target set where the adversarial loss f_A is optimized over, and a test set which is unknown to both the recommender and the adversary—the test set is used to check whether the success of adversarial intent generalizes from the target to the test set. We form the target and test sets in two ways: (E2-a) either using the leave-one-out setup, i.e., leaving one tuple per user in the target set (and another tuple in the test set), or (E2-b) an 80-10-10 split, i.e., splitting the original dataset into 80% of the total ratings per user for training, 10% for the target set, and 10% for the test set.

The recommender R under attack is a matrix factorization (else low rank) model, trained on explicit ratings in the scale $\{0, 1, 2, 3, 4, 5\}$. Unless otherwise specified, we set the latent factor dimension d to 40, the regularization parameter λ to 0.001, and R is trained before the attack for 10 alt-min iterations.

For the adversary A, we set the SVD approximation rank K to 30, and the approximate gradient step constant α to 0.0001. During a single Z-SGD iteration for each of the $K+1$ f_A evaluations, 5 alt-min iterations of R are performed.

We perform warm-start, i.e., for the $t+1$ Z-SGD iteration, R's parameters are initialized from the ones obtained at the end of the alt-min R iterations from the previous t Z-SGD step.

To be consistent with the original movie ratings from the datasets, every time f_A is evaluated, e.g. either during the approximate gradient computation or the loss computation, the Z values are rounded to the closest integers, and get clipped to $[0, 5]$. Also, to ensure that while performing the Z-SGD updates, the fake user distribution does not diverge from the real distribution, we perform *projected* gradient descent:

$$\tilde{Z}_{t+1} = Z_t - \eta \nabla_{Z_t} f_A \tag{7.8}$$

$$Z_{t+1} = \text{clip}(\tilde{Z}_{t+1}, 0, 5), \tag{7.9}$$

where (7.9) corresponds to a box-projection.

To evaluate the success of the adversary, we use various metrics; every time we introduce a new metric, we will use bold letters. Overall, we use the metric of **Attack Difference**, (or else **magnitude of the attack**) denoted by $\Delta(Z)$:

$$\Delta(Z) = f_{\text{before}}(X) - f_A(X; Z) \tag{7.10}$$

where $f_{\text{before}}(X)$ denotes the value of the adversarial loss before Z is concatenated with the real users' data (before the attack). In other words this metric expresses the decrease of the adversarial loss. In order for the attack to be considered successful, $\Delta(Z)$ needs to be at least positive, and ideally larger than zero by a certain margin. Depending on the intent of the adversary, as encoded by f_A, this metric could imply for example a decrease in the predicted score of a target item—in which case, it is related to the "prediction shift metric" [98, 126]—, or a change in the recommendation quality of a target group of users.

Each of the following sections introduces a separate attack type, as specified by target user(s), item(s) and intent of A.

7.4.3 Targeting a User-Item Pair

We start with the adversarial intent: can A learn realistic users that reduce the predicted score for an *unrated user-item entry*? For this, we adopt the (E1) experimental setup.

Let target user be denoted with u and target item with h, where $h \notin \text{RatedBy}(u)$. The adversarial loss is the predicted score for (u, h): $f_A^{(u,h)}(X; Z) = \hat{y}(u, h)$ and the magnitude of the attack is $\Delta^{(u,h)} = f_{\text{before}}^{(u,h)}(X) - f_A^{(u,h)}(X; Z)$, where $f_{\text{before}}^{(u,h)}(X)$ denotes the predicted score for the pair by R before the attack.

We set η to 100, α to 50, K to 5. The adversary performs a total of $T = 21$ Z-SGD iterations, or fewer if a certain stopping criterion is satisfied. We explore two stopping criteria, and two cases for how to specify the (u, h) target entry.

Stopping criterion is $\Delta \geq 1$

First, we considered to stop the Z-SGD iterations when the predicted score for (u, h) is decreased by at least 1 after the attack. We performed this experiment for 70 uniformly at random sampled items for the MovieLens 100K dataset. For each target item h, we sampled target user u from the set of users who have not rated h. Considering an attack successful only if Δ is larger than 0, we found that for only 2 out of the 70 sampled target items the attack was not successful.

Targeting Top Item of User, Stopping criterion: Remove from Top

Next, we used the early-stopping criterion that the item does not exist in the top of the recommendation list anymore, as this aligns better with the actual user experience in a recommendation setting. For this, we randomly sampled target users, and for each user u, we considered as target item h the one out of the unrated set predicted to be at the top of the user's list before the attack—simply put, the top item h of user u. Beyond the attack difference metric 7.10, and the distance metrics (7.6), (7.7), we report the metric of **Rank Loss @ top**$(h) = \mathbb{1}[\text{item } h \text{ @ top-list}]$, where we considered *top* $= \{1, 5, 10\}$.

In the first experiment, for the stopping criterion, we set to remove h from the top-1 list: we found that out of the 135 sampled target users, only for one user the top-one item remained at the top.

In the second experiment, we set for the stopping criterion to remove h from the top-10. We found that out of the 55 sampled users, only for two users the attack was not successful—for the rest, notably, the adversary managed to remove the target item

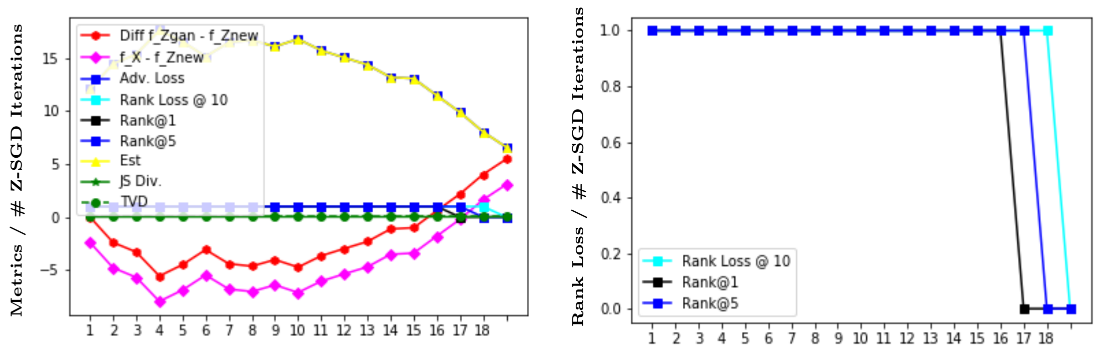


Figure 7.3: The adversary A targets the top-1 item h ID-1062 "A Little Princess (1995)" for user u ID-0, with stopping criterion that h is removed from the top-10 recommendation list of u. x-axis: SGD iterations varying from 1 to 19, with 1 corresponding to using as Z the Z_{GAN}. **Left:** The y-axis represents metrics capturing the success of A in targeting (u, h), and distance metrics for real-fake user distribution. **Right:** Zooming in the ranking loss @$top : \{1, 5, 10\}$. **Gist:** As Z-SGD updates progress, (i) the adversarial loss ('Adv. Loss' f_A, blue line), which is the same as the estimated score of R on the target (u, h) pair ('Est.', yellow line) goes down; (ii) the attack difference $\Delta(Z)$ (magenta line) increases, and the difference of the score $f_A(X; Z_{\text{GAN}}) - f_A(X; Z)$ (red line) increases too; (iii) the rank losses @ top-1/ 5/ 10 (black/ blue/ cyan lines) reach 0 after 17/ 18/ 19 Z-SGD iterations; (iv) the real-fake distribution distance metrics (mean 'JS Div', mean 'TVD', green lines) remain close to 0, despite the adversarial updates.

from the target user's top-10 list, while looking realistic. As an example, Figure 7.3 illustrates the metrics for the movie "A Little Princess" that appeared in the top-1 of user ID-0 before the attack. We can see that the attack is successful, removing the top item from the top-1 at iteration 17, and from top-10 at iteration 19 (thus stopping the Z-SGD updates), optimizing well the adversarial loss of the estimated score for the (u, h) entry (yellow line), while the real-fake distribution distance metrics of mean TVD and mean JS Div (green lines) remain close to 0.

This experiment illustrates our second key finding:

> *The adversary can successfully target the top-1 predicted item of a user, and remove it from the top-10.*

7.4.4 Targeting Item's Mean Predicted Score

Here, we examine whether the adversary can accomplish a more ambitious goal: can A target (push down) the mean predicted score of a target item h over *all* real users

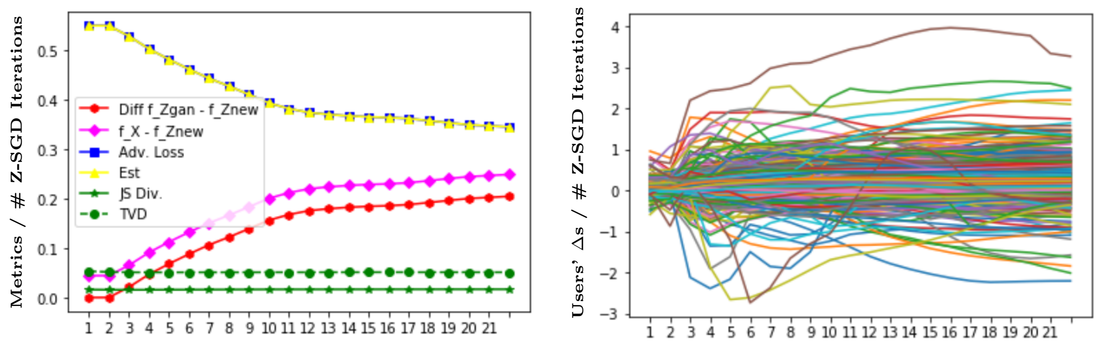


Figure 7.4: The adversary A targets item's h ID-1348 "Mille bolle blu (1993)" average predicted score by R over all users $\{u \in \mathcal{U}\} \notin \mathrm{Ra}(h)$, with stopping criterion that $\Delta(Z) \geq 1$. x-axis: SGD iterations varying from 1 to 22, with 1 corresponding to using as Z the Z_{GAN}. **Left:** The y-axis represents the average metrics over all users. The metrics presented are the same as in Figure 7.3, except for the rank loss metrics which are not included. The 'Adv. Loss'/ 'Est.' is given by (7.11). **Right:** Every line corresponds to each user's $\Delta(Z)$ varying with Z-SGD updates; the average over these lines is the magenta line on the left plot. **Gist:** Although on average the attack can seem successful, i.e., the $\Delta(Z)$ increases and the distance metrics remain close to 0 (left), each user's Δ follows its own trend (right). We conclude that such an attack is difficult.

who have not rated h in the training dataset?—again, we adopt the (E1) experimental setup. The reason for choosing this target user set is because these are the users for which h can be a candidate item for recommendation. This intent can be formulated as:

$$f_A^h = \sum_{u \in \mathcal{U}, \notin \mathrm{Ra}(h)} \hat{y}(u, h), \tag{7.11}$$

and the attack will be successful if $f_{\mathrm{A}}^h(X; Z)$ becomes smaller than the predicted score $f_{\mathrm{before}}^h(X)$.

We keep the same setting as before, except for setting $\eta = 1000$ and choosing α in $\{500, 1000\}$, as we found that for this experiment larger values tend to lead to larger Δ. The early-stopping criterion is $\Delta(Z) \geq 1$. We performed this experiment for 29 randomly chosen target items from MovieLens 100K and found that out of the 29 items, only for 6 items early stopping was realized. Also, for 6 out of the 29 items, the attack was unsuccessful; $\Delta \leq 0$, i.e., the average score after the attack remained the same or increased. Overall, we conclude that:

Targeting the average predicted score of an item is a hard task.

To understand why this happens, we examine how the distribution of Δ over all users who have not rated target item h evolves over the Z-SGD iterations. From Figure 7.4 we can see for the sampled movie "Mille bolle blu (1993)" (similar behavior is noticed in the others too), that although the average difference reached 0.2 (magenta line in Figure 7.4, *left panel*), every user's attack difference follows its own trend (Figure 7.4, *right panel*); with mainly the users with the largest or smallest Δ affecting the average Δ. This shows that the fake users cannot move all users' scores on h *simultaneously* to the same direction.

7.4.5 Targeting the Top User of an Item

In reality, to attack a target item, the adversary does not need to solve the more difficult problem of pushing down *all unrated users*'s score. Instead, they only need to push the score of users who would be good candidates for getting this item in their recommendations. Put differently, these are the users with the higher predicted scores from R before the attack; the rest of the users would not get h in their recommendations either way.

In this experiment, the adversary's intent is to target the *top user* of an item, i.e., the user from $u \notin \mathrm{Ra}(h)$ with the largest predicted score from R before the attack. This can be seen again as a *targeting a single* (u, h) *entry attack*, that we found earlier in Section 7.4.3 to be a successful attack; but while h can be any arbitrary target item, u is the *top user* of the item.

In Figures 7.5, 7.6 we show the results of targeting the top user u of item h "The Joy Luck Club" (similar results hold for other target movies). We can see that just by targeting the predicted score of the most-wanting-"The Joy Luck Club" userID 1417, the scores of all other users who have not rated h get affected too; similarly the scores of userID 1417 for all the other movies he has not rated get affected.

Figure 7.5 shows how the mean attack difference (y-axis) when considering only the top/ bottom users for the item h (*left panel*), or when considering only the top/ bottom items for the user u (*right panel*), varies as we vary the top/ bottom (x-axis).

Figure 7.5, *left panel* shows that although the average difference over all users who have not rated h is only .48, if we consider only the top-5 users with the highest prediction scores for h before the attack, the average difference is 8. In fact, for the top-80 users the average difference is larger than 2, whereas for the users who were predicted

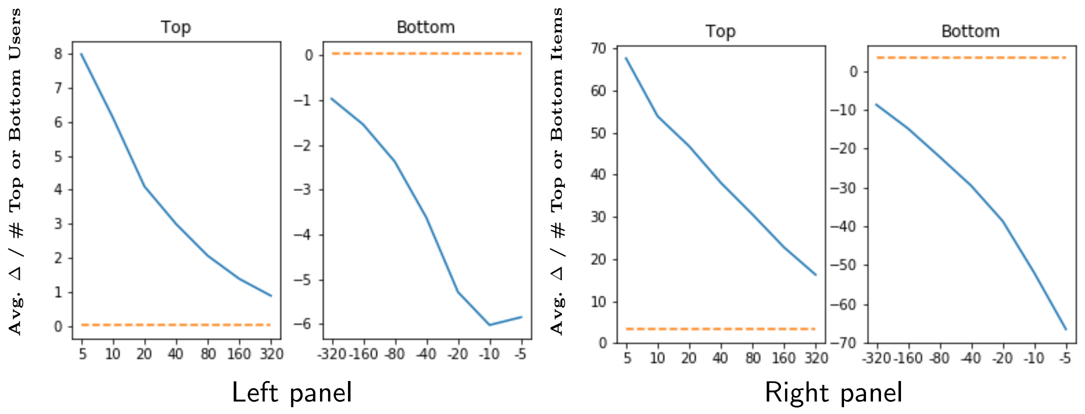


Figure 7.5: **Main Result.** The adversary A targets the *top user* u of item h ID-1417 "The Joy Luck Club" (1993). We say *top/ bottom* to refer to what was predicted with the highest/ lowest score from R before the attack. We focus on the metric of attack difference $\Delta(Z)$ using the Z obtained after the 21 Z-SGD updates, or when stopping criterion $\Delta(Z) \geq 1$ was satisfied; but we define $\Delta(Z)$ over different subsets of users or items. **Left:** x-axis: varying number of users considered to compute $\Delta(Z)$, from 5 to 320 top users, or from 320 to 5 bottom users, for item h. y-axis: The orange dotted line is the average over *all* users $\notin \text{Ra}(h)$, computed to .48. The blue line connects points of the average $\Delta(Z)$ over the corresponding subset of users, e.g., for the top-5 users $\Delta(Z) = 8$, for the bottom-5 users $\Delta(Z) = -6$. **Right:** x-axis: varying number of items out of $\notin \text{RatedBy}(u)$ considered to compute $\Delta(Z)$, from 5 to 320 top items, or from 320 to 5 bottom items for user u. y-axis: Same as left panel, but instead of top/bottom users for h, here Δ is computed over top/ bottom items for u. **Gist:** Targeting the top predicted user of an item attacks also the top-K predicted users of this item, and the top-K predicted items of this user. Also, the bottom predicted users/ items are attacked, as for them the score is increased, which is the opposite from what they would want.

to hate the item the most (bottom-5) the average difference is close to -6; this means that for users who were predicted to least like the item, the predicted score increased, which is opposite to what a good recommender should do.

The *right panel* illustrates a similar story for the items. The average difference on the items which were predicted to be liked the most by u before the attack is > 60 (items which were a good fit for this user are pushed down), and the average difference on the items which were predicted to be liked the least by the user is -70 (items which

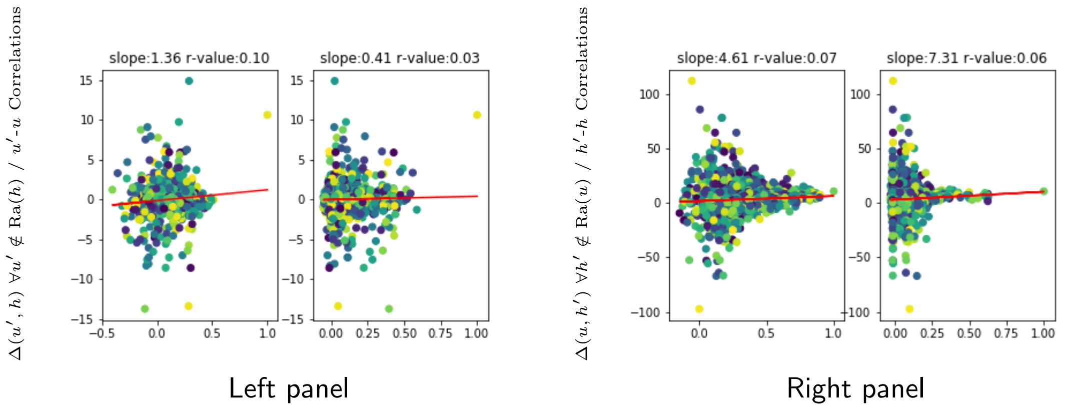


Figure 7.6: Same setup as Figure 7.5. The effect of targeting the top user u for item h ID-1417 on the rest of users who have not rated h; and on the rest of items that have not been rated by the top-user u. For both cases, each dot in the scatter plot (colors are for aesthetic purposes) is the attack difference for a single (user, target item) entry or (target user, item) entry. **Left:** x-axis represents the user-target user correlations, computed either based on the estimated latent factors U before the attack (left), or based on the real ratings, i.e., rows of X (right). **Right:** x-axis represents the item-target item correlations, computed either based on the estimated latent factors V before the attack (left), or based on the real ratings, i.e., columns of X (right). **Gist:** The red line represents a linear-line fitting on the data points to find if there is a relationship between $\Delta(Z)$ and correlations. Although the line-slope is positive, the r^2 is small, thus rejecting the hypothesis for a relationship. However, when only the most correlated ($> .35$ correlation) users are considered, r^2 is .74 for latent-factor based correlations—showing that as the correlations increase, $\Delta(Z)$ also increases. Such a relationship does not hold for the case of items ($r^2 = .29$ for all items with item-item correlation $> .6$); we see though that as item-target item correlations increase, the variance over the Δs reduces.

were a bad fit for the user are pushed up).

This illustrates an important finding of this work:

> *A successful attack on item h is: When A targets the score of the top user, i.e., the user u predicted by R to like h the most before the attack, then the top-K users for h and top-K items for u are also attacked.*

On a side note, we want to see whether there is a relationship among the attack differences Δ of the various users for the target item h with the user-target user u

correlations; or among the Δs of the target user u for the various items with the item-target item h correlations. Thus, Figure 7.6 shows how the different users' (left panel) or items' Δs (right panel) vary as a function of the correlation with the target user or target item respectively. We compute the correlations (a) based on the estimated latent factors U (V) of the R before the attack, or (b) based on the true rating matrix X. We find that for items with small correlation with the target item, the variance of the differences is large; as the correlation increases, the variance reduces. We also find that for the users who are most correlated with the target user (> 0.6 factor-based correlation), the more correlated they are with the target user, the higher the attack difference.

7.4.6 Targeting a Group of Items

Next, we focus on attacks that target an entire *group* of items, in contrast to the presented experiments so far, where a single item was the target of each attack. We examine two adversarial goals:

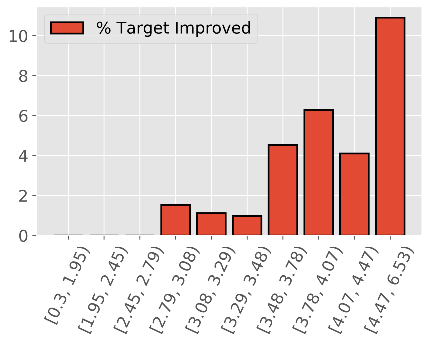
(A1) minimize the mean predicted score over all items in a group, and
(A2) maximize the prediction error, as measured by mean absolute error, over a group.

Experimental Setup. We adopt the (E2) experimental setup (Section 7.4.2) of using the extra information of a "target set"—the adversary A has the added power of, besides making queries to access R's predictions, being able to target some held-out tuples of (user, item, score) which have not been used as part of R's true training data. This is in contrast to the (E1) setup where A targeted one or a set of *unrated* user-item entries.

We used the (E2-b) setup of 80-10-10 split of ratings per user.

To define the target item groups, we explored four different ways: (i) grouping them into 10-percentile groups based on the predicted scores from R before the attack, (ii) using the side information of movie genre, where the same movie can belong to multiple groups, (iii) 10-percentile groups based on the prediction error of R before the attack, or (iv) 10-percentile groups based on number of training ratings per item. To be more precise, for (i) and (iii), we first computed for each item h R's average predicted scores, and average mean absolute error respectively, over the corresponding (user, h, score) tuples in the target set, and we then divided them into deciles based on these values.

Setting. For R, we set d to 100, and λ to 0.1, and we train it for 100 alt-min iterations before the attack. For the adversary A, we set η to 1000, K to 5, α to 50, and $T = 30$.



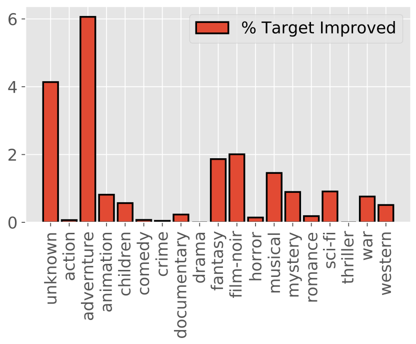


Figure 7.7: The adversary A targets the predicted score–goal (A1)–of a group of items defined over (user, item, score) entries in the target held-out set under the (E2) setup. y-axis: % target improved (percentage of decrease in the f_A). x-axis: target group bins. **Left:** Groups are 10-percentile groups based on R's predicted scores before the attack. **Right:** Groups are defined based on movie genres. **Gist:** % improved is larger for groups with higher predicted score before the attack; up to 6.1% decrease in predicted score happens for the "adventure" genre.

We report the best results from A's side, for the Z-SGD iteration with the best value of f_A in the target set. We report the

$$\textbf{\% Target Improved} = \frac{\Delta * 100}{f_A^{\text{before}}}, \tag{7.12}$$

where Δ is given by (7.10).

Figure 7.7 focuses on goal (A1)—decreasing the average predicted score from R over an item group, defined as the mean of the predicted scores over the subset of the target held-out user-item entries belonging to that group. Figure 7.7, *left panel* shows that A tends to be capable of larger % target improved, i.e., larger % of decrease in the predicted score, for the movie groups with larger original predicted score; A can achieve up to 10.9% for the bucket with predicted scores before the attack between [4.47, 6.53). This is interesting, as these are the entries which would be more likely to appear on users' lists, if the attack did not happen. Figure 7.7, *right panel* shows that when grouping the movies into buckets based on their genre, A can achieve up to 6.1% decrease in predicted score for the adventure genre; the second largest % was found for the unknown genre.

Figure 7.8 focuses on goal (A2)—maximizing the target prediction error of a group. The *left panel* shows the results for grouping movies based on the target prediction error

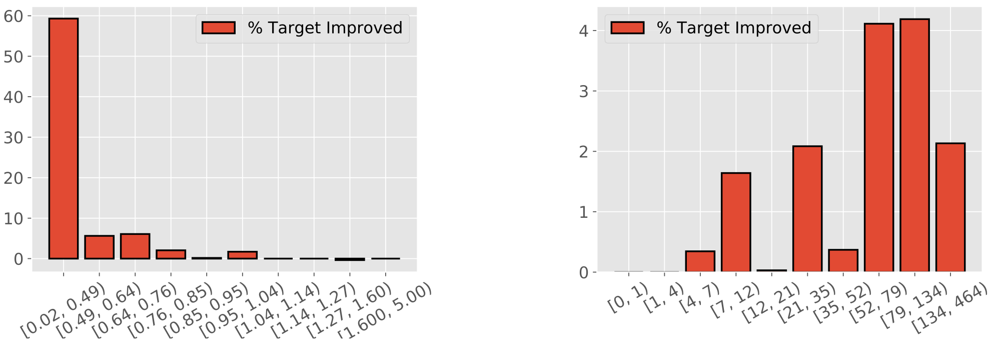


Figure 7.8: The adversary A targets the prediction error–goal (A2)–of a group of items defined over (user, item, score) entries in the target held-out set under the (E2) setup. The x-axis (groups), and y-axis (% Improved) are as described in Figure 7.7. **Left:** Groups are 10-percentile groups based on R's initial, i.e., before the attack, prediction error. **Right:** Groups are 10-percentile groups based on number of training ratings per item. **Gist:** % improved is larger for groups with smaller initial predicted error (left); as number of training ratings per item increase, the attack can be more successful (right).

of R before the attack, and shows that A can achieve up to 59.3% target prediction error increase for the well-modeled buckets, i.e., those with [0.02, 0.49) error before the attack. The *right panel* shows that A can achieve up to 4.2 % error increase for the item bucket with [79, 134) training ratings; however, the groups with fewer than 4 training ratings were not successfully targeted.

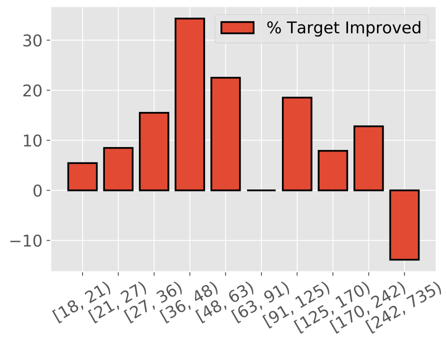
7.4.7 Targeting Improved Modeling for a Group of Users or Items

In the last set of experiments, the adversary's intent is to achieve improvement in the modeling of groups of users or items in the target set—we will refer to those as *targeted improvements*. The difference between this experiment and Section 7.4.6 is that here, A wants to *improve* how R models groups of users or items, whereas before in the (A2) goal A wanted to *deteriorate* how R modeled the groups of items.

We examine three goals:

(I1) improve the average recommendation quality, as measured by Hit Rate@10 (checking whether on average per user, the user's held-out entry is included in their top-10 recommender list (hit=1), or not (hit=0)), over a user group,

(I2) improve the modeling, as measured by mean absolute predicted error, over a group of items, and



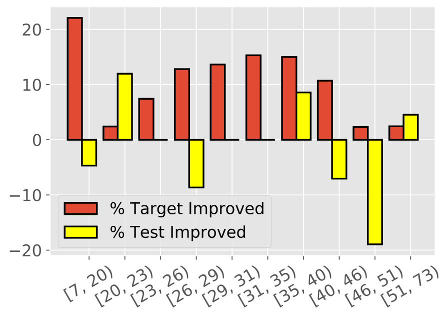


Figure 7.9: The adversary A targets to improve the recommendations, as measured by Hit Rate, of groups of users—goal (I1)—, defined over over (user, item, score) entries in the target set under the (E2) setup. The x-axis (groups), and y-axis (% Improved) are as described in Figure 7.7. **Left:** Groups are 10-percentile groups of users based on number of training ratings per user. **Right:** Groups are 10-percentile groups based on age. **Gist:** The largest improvement is observed for the [36, 48) training ratings bucket (left); for the youngest age bucket (right).

(I3) ensure that two user groups are equally well modeled, i.e., the gap between their modeling errors is reduced.

Goal (I2) is essentially the negative of goal (A2). In all three (I1), (I2), (I3) goals, the metrics are defined in the target set.

The setup followed is again the (E2) experimental setup (Section 7.4.2), and the parameter setting stays the same as described in Section 7.4.6. However, for the sub-experiment focusing on the (I1) goal, the target set is formed based on the (E2-a) leave-one-out setup so to compute the Hit Rates, whereas for (I2), (I3) the (E2-b) 80-10-10 split was used.

We explore different ways of defining the target user or item groups: For the experiment realizing goal (I1) we group users into 10-percentile groups based on number of training ratings per user, or based on the side information of age. For the experiment realizing goal (I2), we create the item groups by grouping movies into 10-percentile groups based on number of training ratings per item. Finally, for the goal (I3)-experiment, we create user groups either based on side information, dividing into two user groups based on gender, i.e., male and female, or dividing them into four 25-percentile groups using the number of training ratings per user.

For the (I1), (I2) set of experiments, we report again the % target improved metric, i.e., the percentage of improvement in the average Hit Rate over the users belonging to

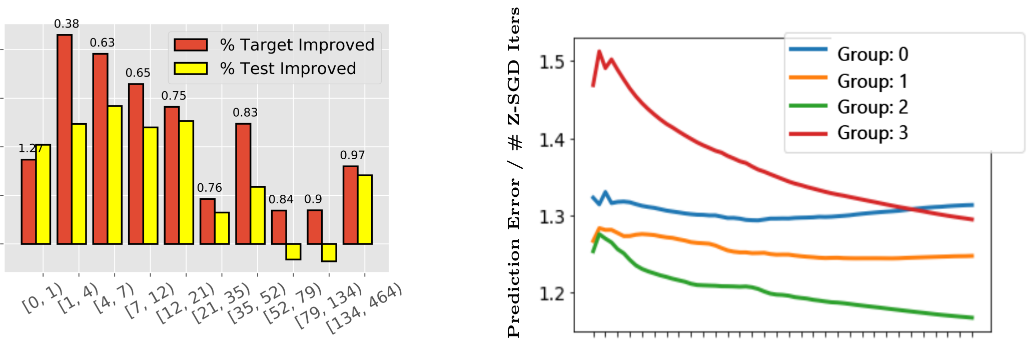


Figure 7.10: The adversary A targets to improve the modeling of groups of users or items; setup is the same as in Figure 7.9, focusing on goals (I2), (I3). **Left:** Goal (I2)—decrease the prediction error for groups of items, where groups are 10-percentile buckets defined based on number of training ratings per item. The x-axis shows the groups, and the annotations on top of the bars are the initial (before the attack) target prediction errors. **Right:** Goal (I3)—minimize the gap betwen the prediction errors of user groups, thus improving the fairness in the modeling of them. User groups are 25-percentile buckets based on number of training ratings per user. Here, the target is to reduce the gap between groups 0 and 3. The figure shows how the prediction error of each group (y-axis) varies as Z-SGD iterations (x-axis) proceed. **Gist:** The fake users can improve the prediction error more for the item groups with smaller initial prediction error (left); the target gap between groups 0 (blue line) and 3 (red line) becomes 0, affecting also the target prediction errors over the other groups (right).

the group, or the percentage of decrease in the mean absolute error over the target set. For (I3), we measure the gap, i.e., absolute difference, among the two prediction errors in the target set we optimize f_A over.

Figure 7.9 focuses on goal (I1), i.e., improving the Hit Rates of certain user groups. From Figure 7.9, *left panel* we find that the user group with the largest number of ratings is not improved, in fact, it is hurt by the fake users. The groups which benefit the most from A tend to be those in the middle of the rating distribution. Figure 7.9, *right panel* shows that grouping users into deciles [7, 20), [20, 23) up to [51, 73), a targeted improvement is possible; with the largest observed for the youngest age bucket. But, these targeted improvements do not transfer to an unseen test set from the same age group (yellow bars). We argue that this happens as the before-the-attack trends of HRs over the age groups in the target and test set differ (Figure 7.11, *right*).

Figure 7.10, *left panel* focuses on goal (I2), and shows the % improvement results in the modeling of item groups defined based on the number of training ratings per item.

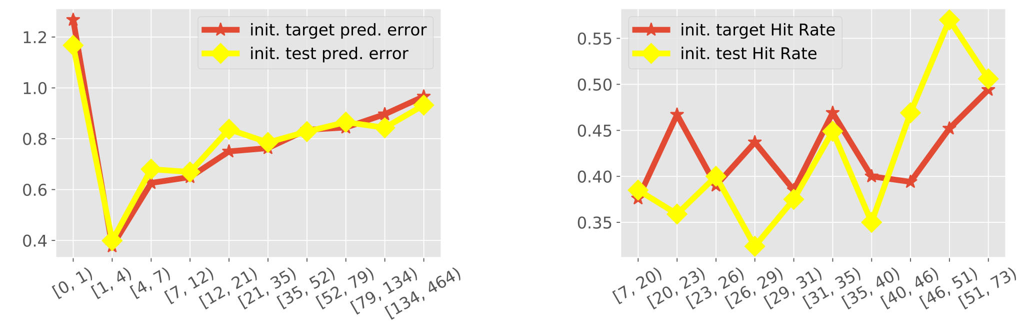


Figure 7.11: **Left:** Initial target and test prediction errors over ratings-based item groups. **Right:** Initial target and test Hit Rates over age-based user groups. **Gist:** When the target-test trends are similar (left), the adversary can target successfully unseen test groups (Figure 7.10, *left*, yellow bars); else when the trends of the initial metrics are different (right), the % Improved trends over target and test are different too (Figure 7.9, *right*, yellow bars).

We find that groups with the smallest target prediction errors before the "attack" (data labels annotated on top of the bars in the plot), are the ones which are better targeted, i.e., the ones with the largest % target improved. Based on the yellow bars, we can also see the % improvements in a test set—we see that in this case the adversary can achieve targeted improvements in an unseen test set, albeit of typically smaller size compared to the respective ones in the target set. From Figure 7.11, *left* we can see that the original metrics of the target and test set hold similar trends across groups, which might be one reason why the attack generalizes here (further future analysis is needed).

Last, we focus on goal (I3), which can be viewed as ensuring *fair treatment* of two user groups. By grouping users into two groups, males and females, and measuring the target mean absolute prediction error, we find that: the error for (females, males) was before the "attack" (1.299, 1.339); during the "attack", A attempts to minimize the error of the group with the larger original error; and after the "attack" the rounded prediction errors became equal: (1.281, 1.281), improving the absolute gap between the two groups from 0.04 to 0.0001 (without the rounding). Similar trends can be found when defining user groups based on number of ratings, or the age side information. For example, Figure 7.10, *right panel* shows that for 25-percentile groups 0, 1, 2, and 3 of users based on number of training ratings per user, the gap between group 0 and 3 becomes 0. Also, the plot shows how targeting 0-3 gap affects the prediction errors of the other groups as well as Z-SGD updates progress—this observation holds for all group-based attacks: targeting a group has an effect of improvement/ decrease in the other groups, too. This inter-connection among the groups seems to play a role for understanding when/ why attacks generalize to a test set (to be explored in the future).

These results serve as proof of concept that:

> *Using fake users generated from A, one can improve in a target set how R models target groups of users or items.*

More rigorous analysis though is needed to understand patterns for the user/ item groups that can enjoy larger targeted improvements.

Discussion

Overall, our experiments indicate that an adversary can achieve a variery of intents on target groups of users or items defined in a variety of ways, while the distance between the real user and fake user distribution remains close[1]. It is important to emphasize again that this happens under the assumptions that the adversary A has the ability to:

- query the recommender R for predicted scores,
- know the underlying true distribution of user-item ratings (so to be able to create realistic-looking user profiles), and
- under the (E2) setup, to target true—not unrated as in the (E1) setup—user-item entries; which however, might not be realizable in real-world settings,

while R is oblivious to A, and is periodically (at every Z-SGD update) warm-start retrained over both real and fake ratings for a few alt-min iterations. Nevertheless, it is still notable that the adversary A can affect the recommender's predictions; with perhaps the most interesting result being that just by targeting the top user predicted for an item, all top users predicted by R for this item, and all top items predicted by R for this user are successfully targeted as well.

7.5 Related Work

Most works on attacking a recommender system R injecting fake users (i.e., "shilling attacks"), have focused on engineering user profiles with high/ small score on the target

[1] Although from Section 7.4.5 onwards we omitted the distance metrics, in all experiments mean TVD and mean JS Divergence remain close to 0.

item(s) and average or normal-distributed scores for (a subset of) the other items. These approaches vary in terms of: the recommender model under attack (e.g. user/ item-based collaborative filtering, model/ memory-based), the adversary's knowledge (e.g. of the true rating distribution, the recommender's architecture or parameters), the attack intent (e.g. promote/ demote an item or group), and the adversary's success metric [11, 34, 120, 127, 126]. Our work is the first where the adversarial fake user generator A *learns* to generate fake profiles in an *end-to-end fashion*, capturing different adversarial intents. Our approach is demonstrated for a low rank R [119], assuming that A knows the true rating distribution, and can evaluate R's objective, but not its gradient.

By learning fake user profiles, we attempt to bridge the gap between shilling attacks and works on adversarial examples [161, 122, 121, 72, 74]. These have largely focused on *classification*; only recently a new adversarial attack type was introduced for graph data [187]. Our approach, although related with works on adversarial examples, has important differences: it (i) consists of injecting fake user profiles during training, instead of perturbing the features of examples on a deployed trained model, and (ii) considers the *recommendation* problem, which can lead to attacks of considerably bigger size than e.g. one-pixel attacks [161], or small-norm perturbation attacks—especially since recommendation models rely on the assumption that similar users tend to like items similarly.

In [55] adversarial classification was formulated in a game-theory framework, giving an optimal classifier given the adversary's optimal strategy. Our work focuses on recommendation and presents the strategy from the adversary's view, considering an oblivious recommender.

Our work is the first to apply generative adversarial nets (GANs) [73], to produce realistic-looking fake users. Previous works have used GANs in recommenders, either for better matrix reconstruction [172], or to find visually similar item recommendations [90]—but never to learn the rating distribution.

When injecting fake user profiles, the original rating data is augmented. Hence, data augmentation works become related [16]. Also, our experiments in Section 7.4.7 make works on adversarial training to tailor representations [66, 63, 24], or on better subset modeling [25, 42, 43] relevant.

Our work does *not* focus on the recommender's strategy to resist; stategies in [130, 71] and robust models [34, 120] remain to be tested. Another direction is to relate our results to stability [2] and ratings' influence [140, 142].

7.6 Summary

In this chapter we presented the first work on *machine learned* adversarial attacks to recommendation systems. We introduced the framework of adversarial recommendation, posed as a game between a low rank recommender R oblivious to the adversary's existence, and an adversary A aiming to generate fake user profiles that are realistic-looking, and optimize some adversarial intent. Our experiments showed that adversarial attacks for a variety of intents are possible, while remaining unnoticeable. A notable attack example is that to ruin the predicted scores of a specific item for users who would have loved or hated that item, it suffices to minimize the predicted score of the top predicted user for that item before the attack.

This study provides several interesting directions for future research on adversarial recommendation. First, our approach needs to be tested in other recommendation datasets, and against other recommendation models, while also varying the knowledge of the adversary A. Second, our work has only scratched the surface on how data augmentation with adversarially learned users can improve the modeling of certain groups; further research is needed to develop a more thorough understanding as to when it could help, and how it would compare with alternative techniques [25, 24]. Third, other optimization objectives could be encoded for the adversarial intent so to craft the recommendations / representations for certain goals [66]. Also, generating new realistic-looking user profiles could be used to improve testbeds of recommendation algorithms. Finally, one of the most important directions is to to create adversary-aware recommenders, and evaluate the degree to which they, as well as existing robust recommenders [34], can resist to such machine learned attacks.

Part IV

Concluding Remarks

Chapter 8

Epilogue

Can we ensure that the very top recommendations that will appear on users' lists will be relevant to them? How can we give good recommendations, while avoiding long queues or out-of-stock messages? Can we create conversational recommender systems, that converse with users to first effectively elicit the preferences—like a human would do—and then give good recommendations? How to best model recommendation as a learning to interact with users problem, while learning on-the-fly the underlying structure among users and items? Is it possible for an external adversarial party, which uses machine learning, to create fake user profiles which cannot be distinguished by real ones, that when inserted in the training of a recommender, can achieve some individualistic purpose?

We argue that these are all important questions which need to be answered, so to lead to a smoother transition towards the next generation of recommendation systems which are fully effective in the real world. In particular, while there has been a lot of progress in the field of recommendation systems, most works have focused on how to improve the quality of recommendations; they usually overlook challenges present in the real-world.

To reduce this gap, this dissertation, combining techniques from machine learning and optimization with real-world insights, offers **novel tools** that capture the real-world challenges while caring for good recommendations, so to **improve the users' personalized experiences**.

Each chapter of this dissertation in turn addresses each of the above questions. In doing so, each chapter gives a novel view of the recommendation process which goes well beyond the classic matrix completion view, popularized after the Netflix challenge.

First, in Chapter 3 we viewed recommendation as a **top ranking problem**, ensuring that by giving a push on relevant items higher ranked than non-relevant ones, the quality of recommendations can be improved (Figure 1.2). Second, in Chapter 4 we saw recommendation as optimizing two objectives: the one of capturing good recommendation quality, and the one of penalizing when the **items' expected usage exceeds capacities**; we found that adding the second objective can make it more unlikely for items to become overcrowded, but also when items' capacities are proportional to user demand, can even improve the quality of recommendations (Figure 1.3).

Both of these views (Part I) can be seen as a special case of how to **encode real-world constraints**—in the first case limited screen space, in the second case limited item capacities—into the loss function. This illustrates the importance of loss functions—how they can correctly guide the learning of the model parameters into better user experience in the real world.

Then, in Part II, we asked the question: Where though does the data used for the learning of the model come from? In this part, a radical shift in viewpoint is made. Instead of assuming that we have already offline data given to us, and we fit our model to this data, so to find associative relationships, we ask the following, from the point of view of the recommendation system: What intervention can I make so to increase the user's recommendation quality? For example, what question can I ask a new user so to (i) learn more about the user, and (ii) narrow down the space of questions? This represents the **novel conversational view** introduced in Chapter 5 (Figure 1.4). Or, for example, what item or list of items can I show to an existing user, so that I exploit what I have learned about the user's preferences so far, while I probe the as-of-yet unexplored areas of the latent preference space? This represents the **learning to interact view**, while using the principle that similar users will probably have similar needs, presented in Chapter 6 (Figure 1.5).

In both of these views, we use the system's underlying uncertainty as a way to guide the exploration. We view the recommendation system as an agent, **making decisions** (which question to ask? which list of items to show?), with the goal of improving the cumulative performance of the algorithm in some fixed time horizon. This is in sharp contrast to the majority of works on recommendation which have focused on static, offline models.

Finally, in the last part (Part III), we thought—what if the data used to train the recommendation system **are not all benign?** We took a rather different perspective from the rest of this thesis; using machine learning to tailor the strategy of an **adversary of the recommender**. In Chapter 7 we saw that it is possible, when the opponent uses machine learning to create indistinguishable user profiles, and to suitably update these profiles to target a certain group of items, or users, to harm the recommendation system (Figure 1.6). This creates an avenue for future work exploring how to resist these machine-learned adversarial attacks.

Taken together, the work presented in this dissertation paves the way towards the next generation of recommendation systems that tackle real-world challenges and put the user's needs first, so to improve the overall user experience. We hope that the presented novel views and corresponding new machine learning models will serve as a stepping stone to how to transition recommendation systems into the wild.

To carefully think how to tailor the objective functions so to capture the constraints present, and lead to an ultimately better user experience.

To view recommender systems as the truly interactive agents they are, and seamlessly learn to converse and generally interact with their users.

To beware that machine learning can be used from an adversarial perspective as well, and thus, the underlying models and learning algorithms need to be robust.

References

[1] Gediminas Adomavicius and YoungOk Kwon. New recommendation techniques for multicriteria rating systems. *IEEE Intelligent Systems*, 22(3), 2007.

[2] Gediminas Adomavicius and Jingjing Zhang. Stability of recommendation algorithms. *TOIS*, 30(4):23, 2012.

[3] Alekh Agarwal, Ofer Dekel, and Lin Xiao. Optimal algorithms for online convex optimization with multi-point bandit feedback. In *COLT*, pages 28–40. Citeseer, 2010.

[4] Deepak Agarwal, Shaunak Chatterjee, Yang Yang, and Liang Zhang. Constrained optimization for homepage relevance. In *WWW*, pages 375–384. ACM, 2015.

[5] Deepak Agarwal and Bee-Chung Chen. Regression-based latent factor models. In *KDD*, pages 19–28. ACM, 2009.

[6] Deepak Agarwal, Bee-Chung Chen, and Pradheep Elango. Explore/exploit schemes for web content optimization. In *ICDM*, pages 1–10, 2009.

[7] Deepak Agarwal, Bee-Chung Chen, Pradheep Elango, and Raghu Ramakrishnan. Content recommendation on web portals. *Communications of the ACM*, 56(6):92–101, 2013.

[8] Deepak Agarwal, Bee-Chung Chen, Pradheep Elango, and Xuanhui Wang. Click shaping to optimize multiple objectives. In *KDD*, pages 132–140. ACM, 2011.

[9] Deepak Agarwal, Bee-Chung Chen, Pradheep Elango, and Xuanhui Wang. Personalized click shaping through lagrangian duality for online recommendation. In *SIGIR*, pages 485–494. ACM, 2012.

[10] Shivani Agarwal. The infinite push: A new support vector ranking algorithm that directly optimizes accuracy at the absolute top of the list. In *SDM*, pages 839–850. SIAM, 2011.

[11] Charu C Aggarwal. Attack-resistant recommender systems. In *Recommender systems handbook*, pages 385–410. Springer, 2016.

[12] Nir Ailon, Zohar Karnin, and Thorsten Joachims. Reducing dueling bandits to cardinal bandits. In *ICML*, pages 856–864, 2014.

[13] Mohammad Aliannejadi and Fabio Crestani. A collaborative ranking model with contextual similarities for venue suggestion.

[14] Kareem Amin, Michael Kearns, Peter Key, and Anton Schwaighofer. Budget optimization for sponsored search: Censored learning in mdps. *arXiv preprint arXiv:1210.4847*, 2012.

[15] George Anders. Alexa, understand me. `https://www.technologyreview.com/s/608571/alexa-understand-me/`, 2017. Accessed: 2017-10-25.

[16] Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*, 2017.

[17] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *JMLR*, 3:397–422, 2003.

[18] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[19] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[20] Suhrid Balakrishnan and Sumit Chopra. Collaborative ranking. In *WSDM*, pages 143–152. ACM, 2012.

[21] Arindam Banerjee. On bayesian bounds. In *ICML*, pages 81–88. ACM, 2006.

[22] Iman Barjasteh, Rana Forsati, Abdol-Hossein Esfahanian, and Hayder Radha. Semi-supervised collaborative ranking with push at top. *arXiv preprint arXiv:1511.05266*, 2015.

[23] Robert M Bell and Yehuda Koren. Lessons from the netflix prize challenge. *ACM SIGKDD Explorations Newsletter*, 9(2):75–79, 2007.

[24] Alex Beutel, Jilin Chen, Zhe Zhao, and Ed H Chi. Data decisions and theoretical implications when adversarially learning fair representations. *arXiv preprint arXiv:1707.00075*, 2017.

[25] Alex Beutel, Ed H Chi, Zhiyuan Cheng, Hubert Pham, and John Anderson. Beyond globally optimal: Focused learning for improved recommendations. In *WWW*, pages 203–212. International World Wide Web Conferences Steering Committee, 2017.

[26] Shalabh Bhatnagar, HL Prasad, and LA Prashanth. *Stochastic recursive algorithms for optimization: simultaneous perturbation methods*, volume 434. Springer, 2012.

[27] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128, 2006.

[28] Stephen Boyd. Alternating direction method of multipliers. In *Talk at NIPS workshop on optimization and machine learning*, 2011.

[29] Stephen Boyd, Corinna Cortes, Mehryar Mohri, and Ana Radovanovic. Accuracy at the top. In *NIPS*, pages 953–961, 2012.

[30] Guy Bresler, George H Chen, and Devavrat Shah. A latent source model for online collaborative filtering. In *NIPS*, pages 3347–3355, 2014.

[31] Sébastien Bubeck, Nicolo Cesa-Bianchi, et al. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, 5(1):1–122, 2012.

[32] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *ICML*, pages 89–96. ACM, 2005.

[33] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11:23–581, 2010.

[34] Robin Burke, Michael P OMahony, and Neil J Hurley. Robust collaborative recommendation. In *Recommender systems handbook*, pages 961–995. Springer, 2015.

[35] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *ICML*, pages 129–136. ACM, 2007.

[36] Nicolo Cesa-Bianchi, Claudio Gentile, and Giovanni Zappella. A gang of bandits. In *NIPS*, pages 737–745, 2013.

[37] Olivier Chapelle and Yi Chang. Yahoo! learning to rank challenge overview. In *Proceedings of the Learning to Rank Challenge*, pages 1–24, 2011.

[38] Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In *NIPS*, pages 2249–2257, 2011.

[39] Harr Chen and David R Karger. Less is more: probabilistic models for retrieving fewer relevant documents. In *SIGIR*, pages 429–436. ACM, 2006.

[40] Li Chen and Pearl Pu. Critiquing-based recommenders: survey and emerging trends. *User Modeling and User-Adapted Interaction*, 22(1-2):125–150, 2012.

[41] Wen Haw CHONG and Ee-Peng Lim. Exploiting contextual information for fine-grained tweet geolocation. In *ICWSM*, pages 488–491. 2017.

[42] Evangelia Christakopoulou and George Karypis. Local item-item models for top-n recommendation. In *RecSys*, pages 67–74. ACM, 2016.

[43] Evangelia Christakopoulou and George Karypis. Local latent space models for top-n recommendation. In *KDD*, pages 1235–1243. ACM, 2018.

[44] Konstantina Christakopoulou. Interactive learning for recommendation. `https://goo.gl/5JLiff`, 2016. AI With The Best.

[45] Konstantina Christakopoulou. Recommendation under constraints. `https://goo.gl/M2ZudM`, 2017. AI With The Best.

[46] Konstantina Christakopoulou and Arindam Banerjee. Collaborative ranking with a push at the top. In *WWW*, pages 205–215. ACM, 2015.

[47] Konstantina Christakopoulou and Arindam Banerjee. Adversarial recommendation: Attack of the learned fake users. *Under Review*, 2018.

[48] Konstantina Christakopoulou and Arindam Banerjee. Learning to interact with users: A collaborative-bandit approach. In *SDM*, pages 612–620. SIAM, 2018.

[49] Konstantina Christakopoulou, Alex Beutel, Rui Li, Sagar Jain, and Ed H Chi. Q&r: A two-stage approach toward interactive recommendation. In *KDD*, pages 139–148. ACM, 2018.

[50] Konstantina Christakopoulou, Jaya Kawale, and Arindam Banerjee. Recommendation with capacity constraints. In *KDD*, pages 1439–1448. ACM, 2017.

[51] Konstantina Christakopoulou, Filip Radlinski, and Katja Hofmann. Towards conversational recommender systems. In *KDD*, pages 815–824. ACM, 2016.

[52] Ingemar J Cox, Matt L Miller, Thomas P Minka, Thomas V Papathomas, and Peter N Yianilos. The bayesian image retrieval system, pichunter: theory, implementation, and psychophysical experiments. *IEEE transactions on image processing*, 9(1):20–37, 2000.

[53] Koby Crammer, Yoram Singer, et al. Pranking with ranking. In *NIPS*, volume 14, pages 641–647, 2001.

[54] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. An experimental comparison of click position-bias models. In *WSDM*, pages 87–94. ACM, 2008.

[55] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al. Adversarial classification. In *KDD*, pages 99–108. ACM, 2004.

[56] Varsha Dani, Thomas P Hayes, and Sham M Kakade. Stochastic linear optimization under bandit feedback. In *COLT*, pages 355–366, 2008.

[57] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW*, pages 271–280. ACM, 2007.

[58] Mark A Davenport, Yaniv Plan, Ewout van den Berg, and Mary Wootters. 1-bit matrix completion. *Information and Inference*, 3(3):189–223, 2014.

[59] Nan Ding, SVN Vishwanathan, Manfred Warmuth, and Vasil S Denchev. T-logistic regression for binary and multiclass classification. *JMLR*, 5:1–55, 2013.

[60] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12(Jul):2121–2159, 2011.

[61] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the l 1-ball for learning in high dimensions. In *ICML*, pages 272–279. ACM, 2008.

[62] John C Duchi, Michael I Jordan, Martin J Wainwright, and Andre Wibisono. Optimal rates for zero-order convex optimization: The power of two function evaluations. *IEEE Transactions on Information Theory*, 61(5):2788–2806, 2015.

[63] Harrison Edwards and Amos Storkey. Censoring representations with an adversary. *arXiv preprint arXiv:1511.05897*, 2015.

[64] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. Developing constraint-based recommenders. In *Recommender systems handbook*, pages 187–215. Springer, 2011.

[65] Daniel Fink. A compendium of conjugate priors. page 46, 1997.

[66] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, Francois Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *JMLR*, 17(1):2096–2030, 2016.

[67] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. Beyond accuracy: evaluating recommender systems by coverage and serendipity. In *RecSys*, pages 257–260. ACM, 2010.

[68] Claudio Gentile, Shuai Li, Purushottam Kar, Alexandros Karatzoglou, Giovanni Zappella, and Evans Etrue. On context-dependent clustering of bandits. In *ICML*, pages 1253–1262, 2017.

[69] Claudio Gentile, Shuai Li, and Giovanni Zappella. Online clustering of bandits. In *ICML*, pages 757–765, 2014.

[70] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[71] Ian Goodfellow, Patrick McDaniel, and Nicolas Papernot. Making machine learning robust against adversarial inputs. *Communications of the ACM*, 61(7):56–66, 2018.

[72] Ian Goodfellow, Nicolas Papernot, Patrick McDaniel, R Feinman, F Faghri, A Matyasko, K Hambardzumyan, YL Juang, A Kurakin, R Sheatsley, et al. cleverhans v0. 1: an adversarial machine learning library. *arXiv preprint*, 2016.

[73] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

[74] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[75] Mark P Graus and Martijn C Willemsen. Improving the user experience during cold start through choice-based preference elicitation. In *RecSys*, pages 273–276. ACM, 2015.

[76] Rupesh Gupta, Guanfeng Liang, Hsiao-Ping Tseng, Ravi Kiran Holur Vijay, Xiaoyu Chen, and Rómer Rosales. Email volume optimization at linkedin. In *KDD*, pages 97–106. ACM, 2016.

[77] LI Hang. A short introduction to learning to rank. *IEICE TRANSACTIONS on Information and Systems*, 94(10):1854–1862, 2011.

[78] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[79] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *TIIS*, 5(4):19, 2016.

[80] Ralf Herbrich. Learning sparse models at scale. In *KDD*, page 407. ACM, 2016.

[81] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, pages 263–272. IEEE, 2008.

[82] Neil Hurley and Mi Zhang. Novelty and diversity in top-n recommendation–analysis and evaluation. *TOIT*, 10(4):14, 2011.

[83] Prateek Jain, Praneeth Netrapalli, and Sujay Sanghavi. Low-rank matrix completion using alternating minimization. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 665–674. ACM, 2013.

[84] Tamas Jambor and Jun Wang. Optimizing multiple objectives in collaborative filtering. In *RecSys*, pages 55–62. ACM, 2010.

[85] Dietmar Jannach, Paul Resnick, Alexander Tuzhilin, and Markus Zanker. Recommender systems—: beyond matrix completion. *Communications of the ACM*, 59(11):94–102, 2016.

[86] Kalervo Järvelin and Jaana Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *SIGIR*, pages 41–48. ACM, 2000.

[87] Christopher C Johnson. Logistic matrix factorization for implicit feedback data. In *NIPS 2014 Workshop on Distributed Machine Learning and Matrix Computations*, 2014.

[88] Mojtaba Kadkhodaie, Konstantina Christakopoulou, Maziar Sanjabi, and Arindam Banerjee. Accelerated alternating direction method of multipliers. In *KDD*, pages 497–506. ACM, 2015.

[89] Daniel Kahneman and Amos Tversky. Prospect theory: An analysis of decision under risk. *Econometrica: Journal of the Econometric Society*, pages 263–291, 1979.

[90] Wang-Cheng Kang, Chen Fang, Zhaowen Wang, and Julian McAuley. Visually-aware fashion recommendation and design with generative image models. In *ICDM*, pages 207–216. IEEE, 2017.

[91] Jaya Kawale, Hung H Bui, Branislav Kveton, Long Tran-Thanh, and Sanjay Chawla. Efficient thompson sampling for online? matrix-factorization recommendation. In *NIPS*, pages 1297–1305, 2015.

[92] Jyrki Kivinen and Manfred K Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–63, 1997.

[93] Nathan Korda, Balázs Szörényi, and Li Shuai. Distributed clustering of linear bandits in peer to peer networks. In *ICML*, pages 1301–1309, 2016.

[94] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[95] Yehuda Koren and Joe Sill. Ordrec: an ordinal model for predicting personalized item rating distributions. In *RecSys*, pages 117–124. ACM, 2011.

[96] Branislav Kveton, Csaba Szepesvari, Zheng Wen, and Azin Ashkan. Cascading bandits: Learning to rank in the cascade model. In *ICML*, pages 767–776, 2015.

[97] Kleanthi Lakiotaki, Nikolaos F Matsatsinis, and Alexis Tsoukias. Multicriteria user modeling in recommender systems. *IEEE Intelligent Systems*, 26(2):64–76, 2011.

[98] Shyong K Lam and John Riedl. Shilling recommender systems for fun and profit. In *WWW*, pages 393–402. ACM, 2004.

[99] Quoc Le and Alexander Smola. Direct optimization of ranking measures. *arXiv preprint arXiv:0704.3359*, 2007.

[100] Joonseok Lee, Samy Bengio, Seungyeon Kim, Guy Lebanon, and Yoram Singer. Local collaborative ranking. In *WWW*, pages 85–96. International World Wide Web Conferences Steering Committee, 2014.

[101] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2014.

[102] Haifang Li and Yingce Xia. Infinitely many-armed bandits with budget constraints. In *AAAI*, pages 2182–2188, 2017.

[103] Huayu Li, Yong Ge, and Hengshu Zhu. Point-of-interest recommendations: Learning potential check-ins from friends. In *KDD*, pages 975–984. ACM, 2016.

[104] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *WWW*, pages 661–670, 2010.

[105] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *WSDM*, pages 297–306, 2011.

[106] Shuai Li, Alexandros Karatzoglou, and Claudio Gentile. Collaborative filtering bandits. In *SIGIR*, 2016.

[107] Xutao Li, Gao Cong, Xiao-Li Li, Tuan-Anh Nguyen Pham, and Shonali Krishnaswamy. Rank-geofm: a ranking based geographical factorization method for point of interest recommendation. In *SIGIR*, pages 433–442. ACM, 2015.

[108] Defu Lian, Cong Zhao, Xing Xie, Guangzhong Sun, Enhong Chen, and Yong Rui. Geomf: joint geographical modeling and matrix factorization for point-of-interest recommendation. In *KDD*, pages 831–840. ACM, 2014.

[109] Dawen Liang, Laurent Charlin, James McInerney, and David M Blei. Modeling user exposure in recommendation. In *WWW*, pages 951–961. ACM, 2016.

[110] Elad Liebman, Maytal Saar-Tsechansky, and Peter Stone. Dj-mc: A reinforcement-learning agent for music playlist recommendation. In *AAMAS*, pages 591–599, 2015.

[111] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[112] Benedikt Loepp, Tim Hussein, and Jüergen Ziegler. Choice-based preference elicitation for collaborative filtering recommender systems. In *CHI*, pages 3085–3094. ACM, 2014.

[113] Tariq Mahmood and Francesco Ricci. Improving recommender systems with adaptive conversational strategies. In *Hypertext*, pages 73–82. ACM, 2009.

[114] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *JMLR*, 11(Jan):19–60, 2010.

[115] Sean M McNee, John Riedl, and Joseph A Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI'06 extended abstracts on Human factors in computing systems*, pages 1097–1101. ACM, 2006.

[116] Aranyak Mehta et al. Online matching and ad allocation. *Foundations and Trends® in Theoretical Computer Science*, 8(4):265–368, 2013.

[117] Tom Minka, John Winn, John Guiver, and David Knowles. Infer .net 2.5. *Microsoft Research Cambridge*, 2012.

[118] Andriy Mnih and Ruslan Salakhutdinov. Probabilistic matrix factorization. In *NIPS*, pages 1257–1264, 2007.

[119] Andriy Mnih and Ruslan R Salakhutdinov. Probabilistic matrix factorization. In *NIPS*, pages 1257–1264, 2008.

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

[121] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. *arXiv preprint arXiv:1610.08401*, 2016.

[122] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *CVPR*, pages 2574–2582, 2016.

[123] Julia Neidhardt, Rainer Schuster, Leonhard Seyfang, and Hannes Werthner. Eliciting the users' unknown preferences. In *RecSys*, pages 309–312. ACM, 2014.

[124] Yurii Nesterov and IU E Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer, 2004.

[125] Trong T Nguyen and Hady W Lauw. Dynamic clustering of contextual multi-armed bandits. In *CIKM*, pages 1959–1962, 2014.

[126] Michael O'Mahony, Neil Hurley, Nicholas Kushmerick, and Guénolé Silvestre. Collaborative recommendation: A robustness analysis. *TOIT*, 4(4):344–377, 2004.

[127] Michael P OMahony, Neil J Hurley, and Guenole CM Silvestre. Promoting recommendations: An attack on collaborative filtering. In *DEXA*, pages 494–503. Springer, 2002.

[128] Pedro A Ortega and Daniel A Braun. Generalized thompson sampling for sequential decision-making and causal inference. *Complex Adaptive Systems Modeling*, 2(1):2, 2014.

[129] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In *ICDM*, pages 502–511. IEEE, 2008.

[130] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. Towards the science of security and privacy in machine learning. *arXiv preprint arXiv:1611.03814*, 2016.

[131] Aditya Parameswaran, Petros Venetis, and Hector Garcia-Molina. Recommendation systems with complex constraints: A course recommendation perspective. *TOIS*, 29(4):20, 2011.

[132] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):123–231, 2013.

[133] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[134] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[135] Filip Radlinski and Nick Craswell. A theoretical framework for conversational search. In *CHIIR*, pages 117–126. ACM, 2017.

[136] Dimitrios Rafailidis and Fabio Crestani. Collaborative ranking with social relationships for top-n recommendations. In *SIGIR*, pages 785–788. ACM, 2016.

[137] Dimitrios Rafailidis and Fabio Crestani. A collaborative ranking model for cross-domain recommendations. In *CIKM*, pages 2263–2266. ACM, 2017.

[138] Dimitrios Rafailidis and Fabio Crestani. Learning to rank with trust and distrust in recommender systems. In *RecSys*, pages 5–13. ACM, 2017.

[139] Alain Rakotomamonjy. Sparse support vector infinite push. *arXiv preprint arXiv:1206.6432*, 2012.

[140] Al Mamunur Rashid, George Karypis, and John Riedl. Influence in ratings-based recommender systems: An algorithm-independent approach. In *SDM*, pages 556–560. SIAM, 2005.

[141] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI*, pages 452–461. AUAI Press, 2009.

[142] Paul Resnick and Rahul Sami. The information cost of manipulation-resistance in recommender systems. In *RecSys*, pages 147–154. ACM, 2008.

[143] Marco Tulio Ribeiro, Anisio Lacerda, Adriano Veloso, and Nivio Ziviani. Pareto-efficient hybridization for multi-objective recommender systems. In *RecSys*, pages 19–26. ACM, 2012.

[144] Jane Ritchie and Jane Lewis. *Qualitative research practice.* Sage Publications, London, 2003.

[145] Neil Rubens, Dain Kaplan, and Masashi Sugiyama. Active learning in recommender systems. In *Recommender systems handbook*, pages 735–767. Springer, 2011.

[146] Cynthia Rudin. The p-norm push: A simple convex ranking algorithm that concentrates at the top of the list. *JMLR*, 10:2233–2271, 2009.

[147] Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *ICML*, pages 880–887, 2008.

[148] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *NIPS*, volume 20, pages 1–8, 2011.

[149] Tim Salimans, Ulrich Paquet, and Thore Graepel. Collaborative learning of preference rankings. In *RecSys*, pages 261–264. ACM, 2012.

[150] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295. ACM, 2001.

[151] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. Recommendations as treatments: Debiasing learning and evaluation. *arXiv preprint arXiv:1602.05352*, 2016.

[152] Steven L Scott. A modern bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry*, 26(6):639–658, 2010.

[153] D Sculley. Large scale learning to rank. In *NIPS 2009 Workshop on Advances in Ranking*, pages 1–6, 2009.

[154] Rajat Sen, Karthikeyan Shanmugam, Murat Kocaoglu, Alexandros G Dimakis, and Sanjay Shakkottai. Latent contextual bandits. *arXiv preprint arXiv:1606.00119*, 2016.

[155] Anna Sepliarskaia, Julia Kiseleva, Filip Radlinski, and Maarten de Rijke. Preference elicitation as an optimization problem. In *RecSys*, 2018.

[156] Hanhuai Shan and Arindam Banerjee. Generalized probabilistic matrix factorizations for collaborative filtering. In *ICDM*, pages 1025–1030. IEEE, 2010.

[157] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, and Alan Hanjalic. Gapfm: Optimal top-n recommendations for graded relevance domains. In *CIKM*, pages 2261–2266. ACM, 2013.

[158] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, and Alan Hanjalic. xclimf: optimizing expected reciprocal rank for data with multiple levels of relevance. In *RecSys*, pages 431–434. ACM, 2013.

[159] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Nuria Oliver, and Alan Hanjalic. Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *RecSys*, pages 139–146. ACM, 2012.

[160] David H Stern, Ralf Herbrich, and Thore Graepel. Matchbox: large scale online bayesian recommendations. In *WWW*, pages 111–120. ACM, 2009.

[161] Jiawei Su, Danilo Vasconcellos Vargas, and Sakurai Kouichi. One pixel attack for fooling deep neural networks. *arXiv preprint arXiv:1710.08864*, 2017.

[162] Mingxuan Sun, Fuxin Li, Joonseok Lee, Ke Zhou, Guy Lebanon, and Hongyuan Zha. Learning multiple-question decision trees for cold-start recommendation. In *WSDM*, pages 445–454. ACM, 2013.

[163] Krysta M Svore, Maksims N Volkovs, and Christopher JC Burges. Learning to rank with multiple objective functions. In *WWW*, pages 367–376. ACM, 2011.

[164] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[165] Liang Tang, Yexi Jiang, Lei Li, Chunqiu Zeng, and Tao Li. Personalized recommendation via parameter-free contextual bandits. In *SIGIR*, pages 323–332, 2015.

[166] Liang Tang, Bo Long, Bee-Chung Chen, and Deepak Agarwal. An empirical study on recommendation with multiple types of feedback. In *KDD*, pages 283–292. ACM, 2016.

[167] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[168] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. In *JMLR*, pages 1453–1484, 2005.

[169] Hastagiri P Vanchinathan, Isidor Nikolic, Fabio De Bona, and Andreas Krause. Explore-exploit in top-n recommender systems via gaussian processes. In *RecSys*, pages 225–232, 2014.

[170] Maksims Volkovs and Richard S Zemel. Collaborative ranking with 17 parameters. In *NIPS*, pages 2294–2302, 2012.

[171] Huazheng Wang, Qingyun Wu, and Hongning Wang. Factorization bandits for interactive recommendation. In *AAAI*, pages 2695–2702, 2017.

[172] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *SIGIR*, pages 515–524. ACM, 2017.

[173] Qing Wang, Chunqiu Zeng, Wubai Zhou, Tao Li, Larisa Shwartz, and Genady Ya Grabarnik. Online interactive collaborative filtering using multi-armed bandit with dependent arms. *arXiv preprint arXiv:1708.03058*, 2017.

[174] Yahoo! Webscope. Dataset, r1.

[175] Markus Weimer, Alexandros Karatzoglou, Quoc Viet Le, and Alex Smola. Maximum margin matrix factorization for collaborative ranking. *NIPS*, 2007.

[176] Max A Woodbury. The stability of out-input matrices. *Chicago, IL*, 93, 1949.

[177] Qingyun Wu, Huazheng Wang, Quanquan Gu, and Hongning Wang. Contextual bandits in a collaborative environment. In *SIGIR*, pages 529–538, 2016.

[178] Jun Xu and Hang Li. Adarank: a boosting algorithm for information retrieval. In *SIGIR*, pages 391–398. ACM, 2007.

[179] Xing Yi, Liangjie Hong, Erheng Zhong, Nanthan Nan Liu, and Suju Rajan. Beyond clicks: dwell time for personalization. In *RecSys*, pages 113–120. ACM, 2014.

[180] Baosheng Yu, Meng Fang, and Dacheng Tao. Linear submodular bandits with a knapsack constraint. In *AAAI*, pages 1380–1386, 2016.

[181] Yonghong Yu and Xingguo Chen. A survey of point-of-interest recommendation in location-based social networks. In *Workshops at AAAI*, 2015.

[182] Jia-Dong Zhang and Chi-Yin Chow. igslr: personalized geo-social location recommendation: a kernel density estimation approach. In *SIGSPATIAL*, pages 334–343. ACM, 2013.

[183] Yongfeng Zhang, Qi Zhao, Yi Zhang, Daniel Friedman, Min Zhang, Yiqun Liu, and Shaoping Ma. Economic recommendation with surplus maximization. In *WWW*, pages 73–83. ACM, 2016.

[184] Xiaoxue Zhao, Weinan Zhang, and Jun Wang. Interactive collaborative filtering. In *CIKM*, pages 1411–1420, 2013.

[185] Masrour Zoghi, Shimon A Whiteson, Maarten de Rijke, and Remi Munos. Relative confidence sampling for efficient on-line ranker evaluation. In *WSDM*, pages 73–82. ACM, 2014.

[186] Shi Zong, Hao Ni, Kenny Sung, Nan Rosemary Ke, Zheng Wen, and Branislav Kveton. Cascading bandits for large-scale recommendation problems. *arXiv preprint arXiv:1603.05359*, 2016.

[187] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *KDD*, pages 2847–2856. ACM, 2018.

Appendix A

Learning to Interact Recommendation Algorithms

A.1 Updates for Learning to Interact Algorithms

Below we give more details into the derivation of the various learning to interact algorithms described in Chapter 6.3.1, with parameter updates summarized in Table A.1. The second, third, and fourth columns correspond to lines 0, 4, and 9 of Algorithms 3 and 4 of Chapter 6.

A.1.1 Independent Bandits

Independent Gaussian Bandits. For a given user i, when $r_{ij} \sim \text{Gaussian}(\theta_{ij}, \sigma_{ij}^2)$, with σ_{ij}^2 known and fixed and unknown mean θ_{ij} for every arm j, we can model the prior distribution of the reward mean θ_{ij} with the Gaussian distribution $\theta_{ij} \sim \mathcal{N}(\mu_{ij}^0, \tau_{ij}^0)$, as it is the conjugate prior to the Gaussian distribution [65]. After incorporating the reward of the item shown $\hat{j}$ for the incoming user i, using Bayes rule we can update the posterior, which will also be a Gaussian due to conjugacy, with mean and variance specified in Table A.1. We refer to this as `Independent Gauss TS`.

Independent Bernoulli Bandits. For a user i, when $r_{ij} \sim \text{Bernoulli}(\pi_{ij})$, it is standard to model the prior distribution of the reward of every arm j with the Beta distribution $\pi_{ij} \sim \text{Beta}(a_{ij}, b_{ij})$, as it is the conjugate prior to the Bernoulli distribution. Due to conjugacy, the posterior will be a Beta distribution $\pi_{ij} \sim \text{Beta}(a_{ij}^{'}, b_{ij}^{'})$ with new parameters $a_{ij}^{'}, b_{ij}^{'}$ defined based on number of observed successes (1) and failures (0). We call this algorithm `Independent Bernoulli TS`; details for each bandit are in [38].

Algorithm	Prior/ Parameter Init.	Posterior	Parameter Update for $\hat{\jmath}$

`Independent`
`Gauss TS`
	$\mathcal{N}(\theta_{ij} \mid \mu^0_{ij}, \tau^0_{ij})$
	Init. μ^0_{ij}, τ^0_{ij}, Input σ^2_{ij}
		$\mathcal{N}(\theta_{ij} \mid \mu^0_{ij}, \tau^0_{ij}, r_{ij})$	$\mu^0_{i\hat{\jmath}} = \frac{\sigma^2_{i\hat{\jmath}} * \mu^0_{i\hat{\jmath}} + \tau^0_{i\hat{\jmath}} * r_{i\hat{\jmath}}}{\tau^0_{i\hat{\jmath}} + \sigma^2_{i\hat{\jmath}}}$
			$\tau^0_{i\hat{\jmath}} = \frac{\tau^0_{i\hat{\jmath}} * \sigma^2_{i\hat{\jmath}}}{\tau^0_{i\hat{\jmath}} + \sigma^2_{i\hat{\jmath}}}$

`Independent` `Bernoulli TS` [38]	$a_{ij} = b_{ij} = 1$ $\text{Beta}(\pi_{ij} \mid a_{ij}, b_{ij})$	$\text{Beta}(\pi_{ij} \mid a_{ij}, b_{ij}, r_{ij})$	$a_{i\hat{\jmath}} = a_{i\hat{\jmath}} + \mathbf{1}[r_{i\hat{\jmath}} = 1]$ $b_{i\hat{\jmath}} = b_{i\hat{\jmath}} + \mathbf{1}[r_{i\hat{\jmath}} = 0]$
`LinTS` [104]	$A_i = I_{d\times d}, \mathbf{b}_i = \mathbf{0}_{d\times 1}$ $\hat{\boldsymbol{\theta}}_i = \mathbf{0}_{d\times 1}$	$\mathcal{N}(\boldsymbol{\theta}_i \mid \hat{\boldsymbol{\theta}}_i, A_i^{-1}, r_{ij})$	$A_i = A_i + x_{\hat{\jmath}} x_{\hat{\jmath}}^T$ $\mathbf{b}_i = b_i + r_{i\hat{\jmath}} \mathbf{x}_{\hat{\jmath}}$, $\hat{\boldsymbol{\theta}}_i = A_i^{-1} \mathbf{b}_i$
`LogTS` [105]	$A_i = I_{d\times d}$ $\hat{\boldsymbol{\theta}}_i = \mathbf{0}_{d\times 1}$	$\approx \mathcal{N}(\boldsymbol{\theta}_i \mid \hat{\boldsymbol{\theta}}_i, A_i^{-1}, r_{ij})$ (Laplace Approx.)	$\pi_{i\hat{\jmath}} = \sigma(\tilde{\boldsymbol{\theta}}_i^T \mathbf{x}_{\hat{\jmath}})$ $A_i = A_i + \pi_{i\hat{\jmath}}(1 - \pi_{i\hat{\jmath}}) \mathbf{x}_{\hat{\jmath}} \mathbf{x}_{\hat{\jmath}}^T$ $\hat{\boldsymbol{\theta}}_i \leftarrow \hat{\boldsymbol{\theta}}_i - \eta(\pi_{i\hat{\jmath}} - r_{i\hat{\jmath}}) \mathbf{x}_{\hat{\jmath}}$
`CluLinTS` [125]	$A_i = I_{d\times d}, \mathbf{b}_i = \mathbf{0}_{d\times 1}$ $\hat{\boldsymbol{\theta}}_i = \mathbf{0}_{d\times 1}$, # clusters	$\mathcal{N}(\boldsymbol{\theta}_c \mid \hat{\boldsymbol{\theta}}_c, A_c^{-1}, r_{ij})$	Update $A_i, \mathbf{b}_i$ based on `LinTS` $A_c = \sum_{i:c} A_i - I$, $\mathbf{b}_c = \sum_{i:c} \mathbf{b}_i$ $\hat{\boldsymbol{\theta}}_c = A_c^{-1} \mathbf{b}_c$
`CluLogTS`	$A_i = I_{d\times d}$ $\hat{\boldsymbol{\theta}}_i = \mathbf{0}_{d\times 1}$, # clusters	$\approx \mathcal{N}(\boldsymbol{\theta}_c \mid \hat{\boldsymbol{\theta}}_c, A_c^{-1}, r_{ij})$ (Laplace Approx.)	Update $A_i, \boldsymbol{\theta}_i$ based on `LogTS` $A_c = \sum_{i:c} A_i - I$, $\hat{\boldsymbol{\theta}}_c = \sum_{i:c} \boldsymbol{\theta}_i$

`Gauss`
`Low Rank TS`
[184, 91]
	$S_{\mathbf{u}_i}^{-1} = \sigma_u^2 \mathbf{I}$
	$S_{\mathbf{v}_j}^{-1} = \sigma_v^2 \mathbf{I}$
	Rand. $\hat{\mathbf{v}}_j$, $\hat{\mathbf{u}}_i$
		$\mathcal{N}(\mathbf{u}_i \mid \hat{\mathbf{u}}_i, S_{\mathbf{u}_i}, r_{ij})$
		$\mathcal{N}(\mathbf{v}_j \mid \hat{\mathbf{v}}_j, S_{\mathbf{v}_j}, r_{ij})$
			$S_{\mathbf{u}_i}^{-1} = S_{\mathbf{u}_i}^{-1} + \tilde{\mathbf{v}}_{\hat{\jmath}} \tilde{\mathbf{v}}_{\hat{\jmath}}^T$, $S_{\mathbf{v}_{\hat{\jmath}}}^{-1} = S_{\mathbf{v}_{\hat{\jmath}}}^{-1} + \tilde{\mathbf{u}}_i \tilde{\mathbf{u}}_i^T$
			$\mathbf{d}_{\hat{\jmath}} = \mathbf{d}_{\hat{\jmath}} + r_{i\hat{\jmath}} \tilde{\mathbf{u}}_i$, $\hat{\mathbf{v}}_{\hat{\jmath}} = S_{\mathbf{v}_{\hat{\jmath}}} \mathbf{d}_{\hat{\jmath}}$
			$\mathbf{b}_i = \mathbf{b}_i + r_{i\hat{\jmath}} \tilde{\mathbf{v}}_{\hat{\jmath}}$, $\hat{\mathbf{u}}_i = S_{\mathbf{u}_i} \mathbf{b}_i$

`Bernoulli`
`Low Rank TS`
	$S_{\mathbf{u}_i}^{-1} = \sigma_u^2 \mathbf{I}$
	$S_{\mathbf{v}_j}^{-1} = \sigma_v^2 \mathbf{I}$
	Rand. $\hat{\mathbf{v}}_j$, $\hat{\mathbf{u}}_i$
		$\approx \mathcal{N}(\mathbf{u}_i \mid \hat{\mathbf{u}}_i, S_{\mathbf{u}_i}, r_{ij})$
		$\approx \mathcal{N}(\mathbf{v}_j \mid \hat{\mathbf{v}}_j, S_{\mathbf{v}_j}, r_{ij})$
		(Laplace Approx.)
			$S_{\mathbf{u}_i}^{-1} = S_{\mathbf{u}_i}^{-1} + \tilde{\pi}_{i\hat{\jmath}}(1 - \tilde{\pi}_{i\hat{\jmath}}) \tilde{\mathbf{v}}_{\hat{\jmath}} \tilde{\mathbf{v}}_{\hat{\jmath}}^T$
			$S_{\mathbf{v}_{\hat{\jmath}}}^{-1} = S_{\mathbf{v}_{\hat{\jmath}}}^{-1} + \tilde{\pi}_{i\hat{\jmath}}(1 - \tilde{\pi}_{i\hat{\jmath}}) \tilde{\mathbf{u}}_i \tilde{\mathbf{u}}_i^T$
			Update $\hat{\mathbf{u}}_i, \hat{\mathbf{v}}_{\hat{\jmath}}$

`Gauss`
`ICF [184]`
	$S_{\mathbf{u}_i}^{-1} = \sigma_v^2 \mathbf{I}$
	Rand. $\hat{\mathbf{u}}_i$
		$\mathcal{N}(\mathbf{u}_i \mid \hat{\mathbf{u}}_i, S_{\mathbf{u}_i}, r_{ij})$	$S_{\mathbf{u}_i}^{-1} = S_{\mathbf{u}_i}^{-1} + \tilde{\mathbf{v}}_{\hat{\jmath}} \tilde{\mathbf{v}}_{\hat{\jmath}}^T$
			$\mathbf{b}_i = \mathbf{b}_i + r_{i\hat{\jmath}} \tilde{\mathbf{v}}_{\hat{\jmath}}$, $\hat{\mathbf{u}}_i = S_{\mathbf{u}_i} \mathbf{b}_i$

`Bernoulli`
`ICF [184]`
	$S_{\mathbf{u}_i}^{-1} = \sigma_u^2 \mathbf{I}$
	Rand. $\hat{\mathbf{u}}_i$
		$\approx \mathcal{N}(\mathbf{u}_i \mid \hat{\mathbf{u}}_i, S_{\mathbf{u}_i}, r_{ij})$
		(Laplace Approx.)
			$S_{\mathbf{u}_i}^{-1} = S_{\mathbf{u}_i}^{-1} + \tilde{\pi}_{i\hat{\jmath}}(1 - \tilde{\pi}_{i\hat{\jmath}}) \mathbf{v}_{\hat{\jmath}} \mathbf{v}_{\hat{\jmath}}^T$
			Update $\hat{\mathbf{u}}_i$

Table A.1: Parametric updates in learning to interact recommendation algorithms. After the feedback $r_{i\hat{\jmath}}$ of user i on shown item $\hat{\jmath}$, the previous posterior becomes the new prior. Time indices are omitted to avoid clutter. Symbol $\tilde{\cdot}$ denotes sampled values.

A.1.2 Contextual Linear Bandits

LinTS. In [104], for a given bandit i, the reward of the arms is assumed to have linear structure: $\mathbf{E}[r_{ij}] = (\boldsymbol{\theta}_i^*)^T \mathbf{x}_j$, where note that only d parameters, typically $\ll N$ need to be learned. `LinTS` assumes that the posterior distribution of $\boldsymbol{\theta}_i$ is $\mathcal{N}(\boldsymbol{\theta}_i \mid \hat{\boldsymbol{\theta}}_i, A_i^{-1})$ where the mean and the variance are updated based on online Bayesian linear regression [27].

LogTS. In [105], Li et al. extended contextual linear bandits for the case when the rewards come from a generalized linear model (e.g. Bernoulli, Poisson, etc.) [27]. Here, we consider only when rewards come from a Bernoulli distribution, where the reward assumption is that $\mathbf{E}[r_{ij}] = \sigma(\boldsymbol{\theta}_i^{*T} \mathbf{x}_j) = 1/(1 + \exp(-\boldsymbol{\theta}_i^{*T} \mathbf{x}_j)$. `LogTS` uses online *logistic* regression to update the underlying parameters. Given that the posterior distribution is not a Gaussian anymore due to the non-linearity introduced, moment matching is employed to approximate it with a Gaussian $\approx \mathcal{N}(\boldsymbol{\theta}_i \mid \hat{\boldsymbol{\theta}}_i, A_i^{-1})$ [27]. Also, as there is no closed form update for $\hat{\boldsymbol{\theta}}_i$, an iterative algorithm is needed.

A.1.3 Contextual Clustering Bandits

CluLinTS [125]. In [125], users are allowed to move to other clusters, based on how close their parameter vector is to the parameter vector of each of the clusters. At every round, after the parameters of the incoming user are updated based on `LinTS`, the cluster parameters are updated, and users are re-assigned to the closest cluster. Briefly, the posterior distribution of $\boldsymbol{\theta}_c$ is a $\mathcal{N}(\hat{\boldsymbol{\theta}}_c, A_c^{-1})$, where the cluster parameters $A_c, \mathbf{b}_c$ are derived by averaging $A_i, \mathbf{b}_i$ of the users $\{i\}$ who are assigned to it (Table A.1, [125]).

CluLogTS. (new method) Here, we devise a new contextual clustered TS algorithm for Bernoulli rewards, referred to as `CluLogTS`, to better capture binary data, such as click/no-click. `CluLogTS` assumes that the reward of item j for user i assigned to cluster c follows: $\mathbf{E}[r_{ij}] = 1/(1 + \exp(-\boldsymbol{\theta}_c^{*^T}\mathbf{x}_j)$. The parameters of the reward distribution of every user i are learned via `LogTS`. Initially, users are randomly assigned to clusters, and cluster parameters are initialized. At every round, an item $\hat{j}$ is shown to the incoming user i based on the sampled parameters of the corresponding cluster c the user is currently assigned to. After incorporating the observed feedback $r_{i\hat{j}}$, user i's `LogTS` parameters get updated, the user is allowed to dynamically move to the closest cluster based on the proximity of the user-cluster parameters, and the clusters' parameters get updated based on the parameters of the users belonging to the cluster.

A.1.4 CF low rank Bandits

ICF. `ICF` [184] was the first work to use CF for recommendation bandits. However, the authors formulated the problem as a contextual linear bandit, where they used as contextual features the latent features of the items as pre-learned from training PMF [148] on some user-item observations. In their formulation, $\mathbf{v}_j$'s are given as input to the algorithm and only $\mathbf{u}_i$'s are learned online. They devised both `Gauss ICF` and `Bernoulli ICF`, with updates found in [184].

`Gauss Low Rank TS`, variants of which have been considered in [91], assumes that the mean reward of the arms follows the low rank assumption, i.e., $\mu_{ij} = \mathbf{u}_i^T\mathbf{v}_j$. In particular, `Gauss Low Rank TS`, using as the underlying model PMF [148], makes the following parametric assumptions for the underlying parameters of the reward distribution at time step t:

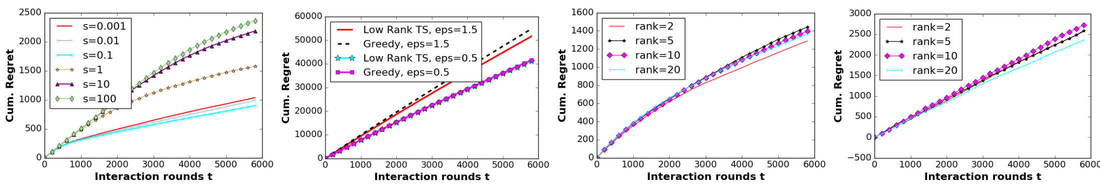
$$\forall i: \ \mathbf{u}_i^t \sim \mathcal{N}(\hat{\mathbf{u}}_i^{(t-1)}, S_{\mathbf{u}_i}^{t-1}) \quad \forall j: \ \mathbf{v}_j^t \sim \mathcal{N}(\hat{\mathbf{v}}_j^{(t-1)}, S_{\mathbf{v}_j}^{t-1})$$
$$\forall ij: \ r_{ij}^t \sim \mathcal{N}(\mathbf{u}_i^{t^T}\mathbf{v}_j^t, \sigma_{ij}^2)$$

where the parametric form of $\hat{\mathbf{u}}_i, \hat{\mathbf{v}}_j, S_{\mathbf{u}_i}, S_{\mathbf{v}_j}$ is defined in Table A.1.

A.2 Sensitivity Analysis

Here, we present a set of experiments to determine the sensitivity of collaborative learning to interact algorithm `Low Rank TS` with respect to different choices of hyper-parameters. For this we consider the following described synthetic data sets which exhibit the CF principle. Specifically, we study how sensitive the algorithm is to (a) the initial parameter setting of the covariance matrices of the latent factors, (b) the noise present in the Gaussian rewards, (c) the rank of the latent factors. For the first two we consider synthetic data, whereas for the last we use the real datasets described in Table 6.1. Following, we describe our synthetic data setup, and then we discuss our sensitivity results illustrated in Figure A.1.

Synthetic Data. For Gaussian rewards, we generated our data via the PMF model, i.e., every entry r_{ij} of the reward matrix R is $r_{ij} = \mathbf{u}_i^T \mathbf{v}_j + \epsilon_{ij}$. For the Bernoulli rewards, we generated our data based on the logistic matrix factorization model: every entry r_{ij} $\sim$ Bernoulli(sigmoid($\mathbf{u}_i^T \mathbf{v}_j$)). For both cases we created different data sets, varying both the number of users M and the number of items N in the range of $\{10, 100, 1000\}$. For every M, N combination, we created five random data sets with different random seeds and report results averaged over them.



(a) Synthetic Data $M = N = 1000$ (b) Synthetic Data $M = N = 1000$ (c) ML100K, Bernoulli (d) Netflix, Gaussian

Figure A.1: (a) Effect of variance s = σ_u, σ_v on `Bernoulli Low Rank TS` for Bernoulli synthetic data with 100 users, 1000 items. Smaller values of s lead to lower R_T. (b) Effect of noise $\sigma_{ij} = \{0.5, 1.5\}$ on `Gauss Low Rank TS` vs. `Low Rank Greedy` for Gauss synthetic data with 1000 users, 1000 items. For higher noise, `Low Rank TS` outperforms `Low Rank Greedy`. (c)-(d) Effect of rank d, varying in $\{2, 5, 10, 20\}$ on `Low Rank TS`.

Effect of σ_u, σ_v for Bernoulli. To determine the effect of the initialization of the covariance matrices $S_{\mathbf{u}_i}, S_{\mathbf{v}_j}$ on `Low Rank TS`, we vary $\sigma_u^2 = \sigma_v^2$ in the range of {0.001, 0.01, 0.1, 1, 10, 100}. We present our results for Bernoulli rewards for the synthetic data set of $M = 100, N = 1000$ in Figure A.1(a). We can see that typically, smaller values of these hyper-parameters lead to lower cumulative regret. We observed a similar trend for the real data sets for both types of rewards.

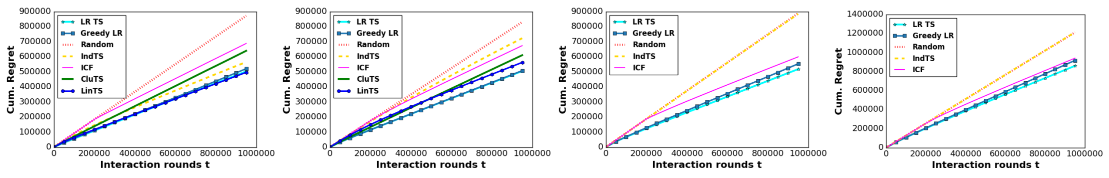
Effect of noise σ_{ij} for Gaussian. Next, we study how the noise level in the user feedback under Gaussian rewards affects `Gauss Low Rank TS` compared to the `Gauss`

`Low Rank Greedy` baseline. We vary the noise standard deviation σ_{ij} to $\{0.5, 1.5\}$ and report the results in Figure A.1(b) for the synthetic data set of 1,000 users interacting with 1,000 items. Similar trends hold for the real data sets too. We can see that when the noise level is small, `Low Rank Greedy` and `Low Rank TS` have similar performance, and the former tends to be better. However, when the noise is larger, TS is more robust and outperforms Greedy. We expect real-world user feedback to be noisy, making Thompson Sampling a possibly better choice.

Effect of Rank. To study the effect of the rank d of the latent factors U, V on the performance of `Low Rank TS`, we vary d in the range of $\{2, 5, 10, 20\}$. Based on Figure A.1, we can see that for Movielens 100K with Bernoulli rewards, rank 2 results in the lowest cumulative regret, while for the data set of Netflix with Gaussian rewards, rank 20 is the best, as the number of observed ratings is an order of magnitude larger. Although in Chapter 6 we present results with rank set to 2, this brief sensitivity experiment shows that a tuned value of the rank would result in even smaller R_T.

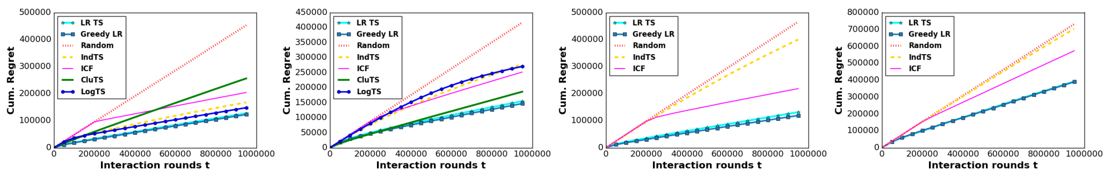
A.3 Real Data Full Experiments

Figures A.2-A.5 contain the experiments on all real datasets, for Bernoulli and Gaussian reward models, for one-item and cascade list recommendation setup.



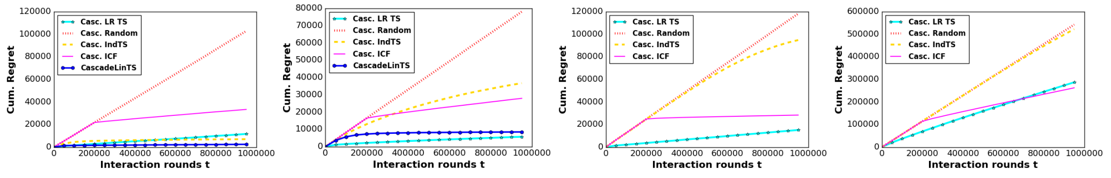
(a) Movielens 100K (b) Movielens 1M (c) Netflix (d) Yahoo

Figure A.2: Comparison of one-item recommendation methods for Gaussian Rewards. `Low Rank (LR) TS` is among the top performing methods for Movielens 100K and Movielens 1M, and outperforms all for larger scale data sets: Netflix and Yahoo.



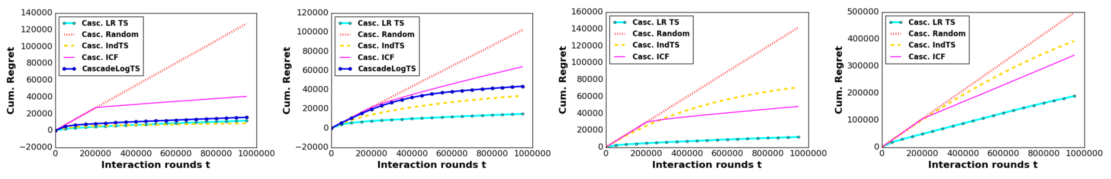
(a) Movielens 100K (b) Movielens 1M (c) Netflix (d) Yahoo

Figure A.3: Comparison of one-item recommendation methods for Bernoulli Rewards. `Low Rank (LR) TS` and `Greedy Low Rank` are the best performing methods.



(a) Movielens 100K (b) Movielens 1M (c) Netflix (d) Yahoo

Figure A.4: Comparison of cascade list recommendation methods for Gaussian rewards. For the larger scale datasets, `Casc. LR TS` is among the top performing methods.



(a) Movielens 100K (b) Movielens 1M (c) Netflix (d) Yahoo

Figure A.5: Comparison of all cascade list recommendation methods for Bernoulli rewards. `Casc. LR TS` achieves the smallest cumulative regret for larger-scale datasets.