User-Centric Design and Evaluation of Online Interactive Recommender Systems

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

Qian Zhao

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Joseph A. Konstan

May, 2018

Acknowledgements

My Ph.D. at the University of Minnesota transformed my way of thinking and changed how I plan to live for the rest of my life. To many people I owe thanks in this process of transformation, which I deeply appreciate. Without their guide, help, patience and love, I would not feel the joy, peace, confidence and hope that I'm full of now.

I am so grateful to my advisor Prof. Joseph Konstan for his advising across my Ph.D. His wisdom, his deep knowledge and his kind support for me to freely explore and think as an independent researcher plays a key role to my transformation. I always feel honored and lucky to be his student. I am thankful for his patience in my mistakes and sometimes stubborn insistence. I will miss the meetings with him where it almost always takes less than 30 seconds before he gives me insightful feedback no matter how horrible my presentation was. I was amazed that the depth of the feedback sometimes took me a year to digest ("Aha! That's what he was talking about one year ago!"). I am also grateful for his kindness to support me and also my family (wife and daughter) so that I can not only pursue a Ph.D. but also have a happy life in these five years.

I am grateful to my wife Tianjiao Jin for being such a great mother taking care of our daughter. I am grateful to my host family Bonnie and Bob Oleson. We had so many sweet and unforgettable memories. They helped me appreciate the stunning beauty of Minnesota. Most importantly, from them, I got to know God's love for the whole world.

I am grateful to Prof. Gedas Adomavicius who spent hours discussing research projects with me and showed me how to think and conduct research rigorously. I am grateful to Prof. Loren Terveen who showed me how to design rigorous scientific experiments in the beginning years establishing a foundation of my Ph.D. research. I am grateful to Research Scientist Dr. F. Max Harper who all the way led me and encouraged me through many of my projects. He taught me how to design and develop

better softwares. I am grateful to Prof. George Karypis who inspired and guided me to think about the sciences of data modeling and computation. I am grateful to Prof. Martijn Willemsen who helped me gain a better understanding on the psychological perspective of recommender system research. I am grateful to Prof. Sashank Varma who guided me in exploring the fields of educational psychology and cognitive sciences. I am grateful to my mentors of several internships, especially Dr. Paul Bennett, Dr. Adam Fourney, Dr. Susan Dumais, Dr. Liangjie Hong, Dr. Shi Yue, Dr. Minmin Chen and Dr. Jilin Chen, who in one way or another shaped my Ph.D. research and my perspectives on the fields.

I first heard about GroupLens from my dear friend and mentor Dr. Zhenhua Dong, who led me into the field of recommender systems. I am grateful to him for his kindness, patience and guidance. I am lucky to be a member of the GroupLens research lab. I felt the warm care and support for each other here as a bigger family. I am grateful for and will miss the numerous times I received detailed and constructive feedback from my labmates. I am grateful to the guidance and help of Dr. Shuo Chang. I am grateful to Raghav Karumur who sit next to me in the lab as a companion and has made the past five years full of fun conversations.

Dedication

To my parents Liangren Zhao and Shuange Wang who unconditionally love me and cultivated the foundation of my personality. To my aunt Suoe Wang and her husband Jingshi Ren. To my aunt Liangmei Zhao. To my uncle Jinwei Zhao and his wife Jiqin Shi. To my uncle Liangsi Zhao. To all my relatives who cared and supported me in Jiexiu, China.

Abstract

User interaction is present in all user interfaces including recommender systems. Understanding user factors in interactive recommender systems is important for achieving better user experience and overall user satisfaction. Many prior works in recommender systems consider recommendation as a content selection process and there is not much prior work focusing on studying user interaction, except user on-boarding interaction design, rating interface design etc. Even for the content selection part, however, it seems obvious that there are a fair amount of factors lying in the scope of user interaction as well, to name a few, visual attention and item exposure, perceived temporal change, reactivity, confusion; *i.e.*, factors regarding content browsing in a typical information system. My research studies several factors while real users are interacting with online recommender systems and answers a series of questions regarding those factors. Specifically, my research focuses on gaining a better understanding on a) whether users pay attention to grids of recommendations displayed in modern recommender interfaces; b) how to interpret and infer user inaction after we show those recommendations to users and further utilize this inaction model to improve recommendation; c) how to organize and present the top-N recommendations to better utilize user attention and increase user engagement; d) how does recommenders optimizing for being engaging (*i.e.*, as many user interactions as possible) affect user experience compared with recommenders optimizing for being right in estimating user preference and maximizing the preference of users on recommendations displayed; e) how to better support work that combines user-centric design, evaluation and building complex, scalable recommendation models going from offline settings into the online environments of providing interactive real-time responses to user recommendation requests, by building a generic recommender server framework.

Contents

Acknowledgements **i**

Dedication **iii**

Abstract **iv**

List of Tables **ix**

List of Figures **xii**

1 Introduction **1**

1.1 Interactivity of Online Recommender Systems 1

1.2 Problems and Approaches 2

1.3 The Research Platform: MovieLens 5

1.4 Thesis Overview 5

1.4.1 Do users see the recommendations? 6

1.4.2 What does user inaction mean? 7

1.4.3 Should we always show the best? 7

1.4.4 Accuracy, engagement or satisfaction? 8

1.4.5 How to support going from offline to online? 9

2 Related Work **10**

2.1 Classical Works in Recommender Systems 10

2.2 Machine Learning Models 12

2.3 User Modeling 15

2.4 User-Centric Evaluation 18
2.5 Recommender Toolkits and Machine Learning Libraries 20

3 Gaze Modeling in Grid-Based Interfaces 21
3.1 Introduction 21
3.2 Related Work 23
3.3 Building Models for the Gaze Prediction Problem 26
3.3.1 Building Linear Models 27
3.3.2 Building Hidden Markov Models 28
3.4 Methods 29
3.4.1 User Browsing Dataset in MovieLens 29
3.4.2 Eye Tracking Protocol Design and Dataset 30
3.4.3 Evaluation 31
3.5 Results 31
3.6 Discussion 35

4 Interpreting User Inaction Feedback 38
4.1 Introduction 38
4.2 Related Work 40
4.3 Data Collection 43
4.4 Interpreting User Inaction 48
4.5 Future Recommendation 48
4.6 Classifying User Inaction 49
4.7 Improve Recommendation 55
4.8 Discussion 56

5 Cycling and Serpentining of Top-N Lists 59
5.1 Introduction 59
5.2 Related Work 61
5.3 Experiment Design 64
5.3.1 Measurements 68
5.4 Results 70
5.5 Discussion 73

6 Optimizing for User Interaction **78**
6.1 Introduction . . . 78
6.2 Background of User-Centric Research . . . 81
6.3 Background of Machine Learning . . . 82
6.3.1 Empirical Risk Minimization . . . 82
6.3.2 Matrix Factorization . . . 83
6.3.3 Q Learning . . . 84
6.3.4 Regret Minimization . . . 85
6.3.5 Contextual Bandit and LinearUCB . . . 86
6.4 Method . . . 87
6.4.1 The Six Recommenders . . . 88
6.4.2 Objective Measurements . . . 93
6.4.3 Subjective Measurements . . . 94
6.5 Results . . . 95
6.5.1 Measurements Interpretation . . . 97
6.5.2 RQ1 . . . 97
6.5.3 RQ2 . . . 98
6.6 Discussion . . . 99

7 A Generic Recommender Server **101**
7.1 Introduction . . . 101
7.2 Related Work . . . 103
7.3 The Generic Server Design . . . 105
7.3.1 Recommender Components and Extensibility . . . 107
7.3.2 Server Interface, Architecture and Scalability . . . 109
7.3.3 Using the Server . . . 112
7.4 Case Studies . . . 115
7.4.1 Extension and Integration . . . 115
7.4.2 Online Recommender Blending . . . 118
7.4.3 System-Level Cold-Start . . . 121
7.5 Discussion . . . 122

8 Conclusion **124**

8.1 Contribution 126

8.2 Future Work 128

8.3 Implication 130

References **131**

List of Tables

1.1 A toy example of the data that a movie recommender system has. . . . 3

3.1 Different models for the gaze prediction problem, in which *bl* denotes baseline, *ub* denotes training only on user browsing data (or *training without fixation*) and *et* denotes training on eye tracking data (or *training with fixation*) as well. . . . 37

4.1 The confusion matrix of the inaction model (rows are the predicted classes and columns are the actual classes) and the accuracy in terms of AUC for each class (binary classification of one vs. others using the probabilistic output of the 7-class classification model). . . . 51

4.2 Three possible ways to utilize the user inaction model in recommender systems. MF denotes Matrix Factorization and FM denotes Factorization Machine. . . . 51

4.3 Coefficients (with standard errors) of predictors from a multinomial regression model predicting the category of user inaction. They represent the log odd-ratio change with respect to the baseline category NotNoticed. "closest" denotes the closest item that has an action if there is any action on the page. "row" or "col" denotes the row or column index of the position in the grid. "Sim" denotes similarity score. "numShow" denotes the total number of displays while "numFrontShow" denotes the total number of *front-page* displays. "[action]Ratio" denotes the ratio of items having the corresponding [action]. *closetInvDist* represents the inverse of the Euclidean distance between the acted-upon item (if there is any) and the inaction item. Significance levels: *p<0.05, **p<0.01, ***p<0.001. . . . 58

5.1 Page-level Column First and Item-level Row First serpentining (PCF-IRF) algorithm illustrated with (top-)N=240, M (number of rows)=4. 1st, 2nd, kth are the page indices. M controls how scattered the new top-N list is in the original rankings and also how much change an item's ranking can have after cycling). 66
5.2 SSA and EXP metrics and their corresponding survey questions. 69
5.3 Condition naming for the interaction between cycling and serpentining factors. 69
5.4 Results of different conditions for INT metrics. *au* indicates the measurement and effect across all users in that treatment group, including users who opt-out, returning to their default recommender. *su* indicates the measurement and effect for users who retain the experimental recommender in the measured first half month. We include both to estimate the effects both on those who retain the treatment and on the population of users offered the treatment overall. We analyzed users who opt-out separately, and in no measurement did they differ significantly from the control group. *numSessions* (overall mean is 4.59) is not shown because there were no statistically significant differences. See Table 5.3 for the definition of condition names that combine *cycling* and *serpentining*. The numbers are means with standard errors in the parentheses and only significant comparisons (through *negative binomial regressions*) are marked with significance codes: + ($p < 0.1$), * ($p < 0.05$), ** ($p < 0.01$). 76
5.5 Results of different conditions for SSA and EXP metrics. See Table 5.3 for the definition of condition names that combine *cycling* and *serpentining*. ctrl. condition is the base to compared with in the ordinal regressions. The numbers are coefficients (in log odd-ratio scale) with standard errors in the parentheses. Significance codes: + ($p < 0.1$), * ($p < 0.05$), ** ($p < 0.01$). 77
5.6 The coefficients and standard errors (in parentheses) of the ordinal regressions using individual SSA to predict EXP (usefulness and satisfaction). Significance codes: + ($p < 0.1$), * ($p < 0.05$), ** ($p < 0.01$), *** ($p < 0.001$). 77
6.1 The six recommenders studied in this work 89

6.2 The coefficients (and standard errors in the parentheses) of the activity metrics predicting user overall satisfaction. These estimates come from an ordinal regression (cumulative link) model treating *satisfaction* response as ordinal values. Each observation is a user who has answered the *satisfaction* question for at least once. 95
6.3 The coefficients (and standard errors in the parentheses) of the perception metrics predicting user overall satisfaction. These estimates come from an ordinal regression (cumulative link) model treating *satisfaction* response as ordinal values while others as continuous values. Each observation is a user who has completed all the survey questions for at least once. . . . 95
6.4 Means of both objective and subjective metrics for the six recommenders. Significant differences after correction are marked with *. Metrics that are not presented here do not show significant differences. The numbers in parentheses for #frontView and #exploreView are standard errors. These estimates come from two negative binomial regression models using MF-Rating as the baseline. The intervals for frontPositive rate, front-Negative rate, exploreNegative rate are 95% confidence intervals. These estimates come three mixed-effects logistic regression models using MF-Rating as the baseline (treating user ID as a random intercept effect). The means of *balance of recency* and *satisfaction* are calculated treating the survey responses as continuous values. The coefficients in the parentheses come from a ordinal regression (cumulative link) model using MF-Rating as the baseline (treating responses as ordinal values). 96
7.1 A high-level comparison of the available softwares for building recommender systems. 105

List of Figures

1.1 The *front* page of MovieLens. 4
1.2 The *explore* page of MovieLens. 4
3.1 In this work, we predict user gaze in grid-based user interfaces. Above are four such layouts – YouTube (top-left), Hulu (top-right), Google Apps (bottom-left) and MovieLens (bottom-right). 22
3.2 Graphical representation of a HMM in which F denotes fixation variable and A denotes action variable. α, T and E are parameters representing categorical conditional distributions defining the HMM. N is the length of the HMM sequence. For the gaze prediction problem, HMM gives inferred probability distributions of F when the values of A are observed. 28
3.3 AUC boxplots for different models in predicting fixation probability. Higher scores are better. See Table 3.1 for descriptions of the models. 32
3.4 MAE boxplots for different models in predicting fixation time. Lower scores are better. See Table 3.1 for descriptions of the models. 32
3.5 ROCs in classifying displayed movie cards into being fixated or not in a page view. It is from one run of the evaluation procedure. 33
3.6 Fitted probabilities for different positions and the effects plot of *position* feature and *dwell time* in predicting whether a displayed movie is fixated using logistic regression. No significant interaction is found. 34
3.7 AUC boxplots of the best model *et:eyeTrackingHmm* in predicting fixation probability for the different tasks. 35
3.8 MAE boxplots of the best model *et:linearModelActionDist* in predicting fixation time for the different tasks. 36

4.1 The flow of the inaction survey illustrating how the questions were asked. N is the number of responses to the corresponding question. The ratios are the proportions of options within those response. 45
4.2 The distribution of future recommendation preference for different user inaction categories. The order of preference (significant after Bonferroni correction, p<0083) is *WouldNotEnjoy* < *Watched* < *NotNotice* < *NotNow or Others Better* < *ExploreLater or DecidedToWatch.* 50
5.1 Illustration of how *cycling* works with an example. 67
5.2 Illustration of how *serpentining* works with an example. 67
6.1 An example illustration of utilizing delayed user feedback *user return* as the learning signal. 90
6.2 The Q-learning curve in training the reinforcement delayed reward model using one year of historical data. The y-axis is the sum of Q-values for all $t = 1, ..., T$ in each epoch on the x-axis. 92
7.1 The environment that the generic recommender server is designed for. . 105
7.2 Samantha Data and Request Processing Flows. The directions of the arrows represent data flow and component dependencies. 110

Chapter 1

Introduction

Recommender systems have been traditionally used for marketing, *e.g.*, Amazon product recommendation, but nowadays are closely integrated into various online information systems that we rely on in our daily life, *e.g.*, personalized search engines, Netflix movie recommendation, maps with place or restaurant recommendation, Facebook friends feed recommendation and personal intelligent assistants (such as Microsoft Cortana, Amazon Alexa etc.) that provide proactive informational assistance for people's current or future tasks. From a broad perspective of the field, the research of these information recommendation systems involves not only modeling user preferences and predicting the next set of items that users might be interested in, but also involves studying the dynamic contexts and interactive usage of people and how such systems can take account of important user factors to design a delightful user experience.

1.1 Interactivity of Online Recommender Systems

Take Youtube recommender systems as an example. When a user goes to the Youtube home page, a grid of recommended videos for the user to watch are displayed. A grid here refers to the layout of the interface in which there are several rows and columns of videos showing information of pictures and titles etc. The user visually examines the displayed videos and maybe gets interested in one of the recommendations and clicks to go to the detail page of the video to watch it. After finishing the video, the user goes back to the home page and realizes that the grid of recommendations have changed based on

the just watched video. The user might not like the new set of recommendations and therefore clicks "show more" link to ask for more recommendations. The user browses another several sets of videos until the user finds another one that is worth watching.

User interaction is present in all user interfaces including recommender systems as illustrated in the above example. Generally speaking, understanding user factors in interactive recommender systems is important for achieving better user experience and overall user satisfaction. Recommender systems are mostly considered as a content selection process in many works of recommender system research and there is not much prior work focusing on user interaction, except, *e.g.*, the user on-boarding process [1] and rating interface design [2] etc. However, even for the content selection part, the above example demonstrates that there are a fair amount of factors lying in the scope of user interaction as well, to name a few, visual attention and item exposure, perceived temporal change, reactivity, confusion; *i.e.*, factors regarding content browsing in a typical information system.

1.2 Problems and Approaches

Classical approaches to recommender systems simplifies the recommendation problem into a preference estimation or item ranking problem, *e.g.*, the classical problems of rating prediction and top-N recommendation.

In the preference estimation approach, the way to make recommendations is to estimate how much a user will like each of the items in the system based on what the system knows about the user's preference on a subset of items. The evaluation of the approach can be converted to measuring how accurate we can do in terms of predicting for unobserved user-item pairs. Imagine a system where there are a few users and items as Table 1.1 shows. The numbers in the table are ratings the system has collected from users on some items. The question marks are unobserved ratings that the system wants to predict. If we can build a model with certain confidence in predicting the unobserved ratings, then we can simply sort items based on the predicted ratings and recommend the top ones that are not observed, yet. This type of approaches are evaluated with the rating prediction accuracy metrics, *e.g.*, the errors of the rating predictions with metrics of Mean Average Precision (MAE) or RMSE (Root Mean Squared Error).

Table 1.1: A toy example of the data that a movie recommender system has.

User	Zootopia	Inside Out	Coco	Black Swan	It
Qian	?	4	?	3	1
Jimmy	2	?	1	?	?
Anna	5	4	?	?	2
Olivia	1	3	2	?	3

From the top-N recommendation perspective, the recommendation problem is considered as selecting the best set of items for a user so that it contains the most items that this user might potentially like. The metrics to evaluate these types of approaches is to compute how accurate an algorithm can rank items that users like to the top (*i.e.*, precision) and how complete the ranked list is in finding all items that users might like (*i.e.*, recall). Widely used metrics are Mean Average Precision (MAP) [3], Normalized Discounted Cumulative Gains (nDCG) [4, 3] etc. Recommendation based on rating prediction can be evaluated in this way as well but the difference is that rating prediction evaluation or preference estimation evaluation takes all items into account including those that users tell us that they do not like, which in real systems may not matter as much to a user since the user might only see the top of the list anyway. Top-N recommendation is particularly suitable for evaluating recommender systems relying on user implicit feedback where the information it collects from users are not real values of how much the user likes an item, but a binary action of whether or not the user consumes or interacts with an item.

However, either the evaluation of preference estimation or top-N recommendation can not capture many aspects of user interaction in recommender systems. For example, it does not answer the question of how we should present the top-N recommendations to users so that they can better process the information in the recommendations. It does not address the issue of whether we should keep showing a recommendation when a user goes back to the home page of the system within the same visit or across multiple visits. The answers to these questions require going beyond offline assumption or modeling into online environments to study how users perceive recommendations while they are interacting the recommendations.

Because of lack of access to production systems and real users, many research works

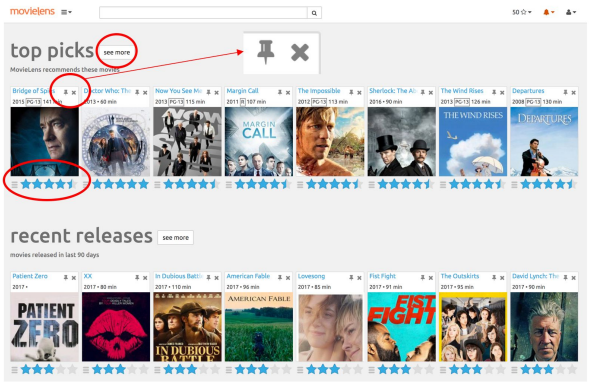

Figure 1.1: The *front* page of MovieLens.

Figure 1.2: The *explore* page of MovieLens.

in recommender systems fall short of looking at user interaction and user experience, instead mostly focusing on offline algorithm innovation. Industrial recommender research on the other hand tends to separate out recommender systems from User eXperience (UX) research mostly treating recommendation as a machine learning problem. This hinders comprehensive user and algorithm understanding. With access to a live system MovieLens that has thousands of active users every month, my research focuses answering questions regarding the user-system interaction process in recommender systems.

1.3 The Research Platform: MovieLens

MovieLens [1] is a movie information site in which users can browse movie information and get personalized movie recommendations. It has thousands of active users every month. In the *front* (or *home*) page of MovieLens, as shown in Figure 1.1, there are several sections displaying movies according to different criteria (*e.g.*, *recent releases*, *most popular* etc.). The top first section is *top-picks*, *i.e.*, according to the criteria of recommendation scores generated by recommendation algorithms. This section has eight movie cards horizontally displayed in standard PC screens (*i.e.*, one row with eight columns). Users can click a "see more" link besides the section to see more top picks in a different type of *explore* pages. Each *explore* page is organized in a three-by-eight grid (*i.e.*, 24 movie cards) and users can browse them page by page, as shown in Figure 1.2.

Each movie card enables three major features: rating, clicking and adding into the wishlist. If a user knows the movie, the user can enter a rating through a five-star rating widget (half-star unit) attached with the movie card to tell the system his or her preference on the movie (this can help the system make better future recommendations). If a user wants to know more about the movie, the user can click the movie card and transition to a page with detailed information (*e.g.*, the plot, cast, trailers). If a user wants to collect the movie for later watching, the user can add the movie into his or her personal wishlist through a wishlist icon on the movie card.

1.4 Thesis Overview

Following the user-centric approach, my thesis focuses on answering the following series of questions regarding the online user-system interaction process in recommender systems, i.e.:

- Do users pay attention to all the displayed recommendations in a typical grid-based recommender interface? How well can we predict user attention on the grid of recommendations?

[1] https://movielens.org

- How to interpret user inaction? How well can we infer the categories of user inaction reasons? How to utilize the inferred category of user inaction?
- Should we always present the top-N recommendations from best to worst? How does cycling and serpentining top-N item lists affect user experience?
- How does treating user interaction or engagement as the objective to optimize for in recommender systems affect user experience, compared with optimizing for the traditional accuracy of user preference estimation? How to better optimize for user interaction or engagement?
- With user experience as the system goal, what type of software tools are possibly needed to accelerate the study of user-system interaction in recommender systems, especially for those based on complex learning algorithms?

1.4.1 Do users see the recommendations?

In Chapter 3, we studied user visual attention on recommendations presented in a grid-based interface. As users browse a recommender system, they systematically consider or skip over much of the displayed content. It seems obvious that these eye gaze patterns contain a rich signal concerning these users preferences. However, because eye tracking data is not available to most recommender systems, these signals are not widely incorporated into personalization models. We show that it is possible to predict gaze by combining easily-collected user browsing data with eye tracking data from a small number of users in a grid-based recommender interface. Our technique is able to leverage a small amount of eye tracking data to infer gaze patterns for other users. Our results show that incorporating eye tracking data from a small number of users significantly boosts accuracy as compared with only using browsing data, even though the eye-tracked users are different from the testing users (*e.g.*, AUC=0.823 vs. 0.693 in predicting whether a user will fixate on an item). We also demonstrate that Hidden Markov Models (HMMs) can be applied in this setting; they are better than linear models in predicting fixation probability and capturing the interface regularity through Bayesian inference (AUC=0.823 vs. 0.757).

1.4.2 What does user inaction mean?

In Chapter 4, we study the possible interpretations of user inaction in recommender systems by deploying a field survey and designing models to infer the category of the reasons of user inaction. Temporally, users browse and interact with items in recommender systems. However, for most systems, the majority of the displayed items do not attract any action from users. In other words, the user-system interaction process includes three aspects: browsing, action and inaction. Prior recommender systems literature has focused more on actions than browsing or inaction. We deployed a field survey in MovieLens to interpret what inaction means from both the user and the system's perspectives guided by psychological theories of human decision making. We further systematically study factors to infer the reasons of user inaction and demonstrate with offline data sets that this descriptive and predictive inaction model can provide benefits for recommender systems in terms of both action prediction and recommendation timing.

1.4.3 Should we always show the best?

In Chapter 5, we challenge the conventional wisdom of recommender systems that optimizes to present a top-N list reflective of the most-highly recommended items a user has not yet rated or consumed. First, top-N does not consider whether a recommendation has already been displayed to the user before, that is, whether it is fresh vs. potentially stale. Second, presenting the standard top-N list may create an experience where continued exploration results in a sense of finding ever-worse alternatives recommended (*e.g.*, when designing a live show, we would not put all the highlights at the beginning and none at the end, because people are not motivated to continue with that design). We conducted an online field experiment to test two ways of manipulating top-N: *cycling* and *serpentining*. Cycling demotes recommended items after they have been viewed several times, while promoting fresher recommendations from the lower portions of the list. Serpentining displays a "zig-zag" order, in which the best recommendations are spread across several pages, offering some high-quality and some lower-quality items on each page as a user continues to explore. We found that both cycling and serpentining increased user engagement to a substantially higher level, *e.g.*, 57.9% to 64.3% increase in the average number of interest-showing actions (*e.g.*, clicks, wishlist additions etc.).

Along with the benefit of increased engagement however, we also observed a trade-off between serving users who are open to deeper interaction or engagement versus those who want efficient recommendations right away, *i.e.*, we found that both cycling and serpentining had negative effects on user subjective perception of recommendation accuracy, familiarity or usefulness and when the cycling change is too much (*i.e.*, cycling every time users go back to the home page of the system), it incurs a significantly higher opt-out rate (users have the option to stop using the assigned recommender condition).

1.4.4 Accuracy, engagement or satisfaction?

In Chapter 6, we studied the effects of a recommender system optimizing for user interaction comparing with a recommender that optimizes for preference estimation accuracy on user experience including user engagement, perception and satisfaction. In the history of recommender system research, there was a transition of trend from modeling user explicit feedback data (*e.g.*, five-star ratings on items) to implicit feedback data (*e.g.*, clicks, watching). Hypothetically, explicit feedback from users carries more precise signals about how much users like an item while implicit feedback from users reflects the users engaging interest or intention towards an item at the time of interaction. However, until today, the difference between these two types of recommender systems built on explicit vs. implicit feedback data is not addressed in the research literature. We set out to compare the two in a live experiment. We showed that a recommender maximizing the likeliness of a user interacting with an item (*e.g.*, clicking to see details or adding into wishlists etc.; representing user implicit feedback) engages users more than a recommender maximizing the predicted ratings of a user on items (*i.e.*, the five-star ratings; representing user explicit feedback). However, we also find the implicit action based recommender is not as precise as the explicit rating based recommender, *i.e.*, although it increased positive user engagement in the system (*e.g.*, high ratings, clicks etc.), it also increased negative user engagement substantially, *e.g.*, low ratings, giving not-interested feedback on recommendations, and significantly higher user browsing effort (which is shown negatively correlated with user satisfaction in the study). We demonstrated that blending both explicit and implicit feedback from users through online interactive learning algorithms gained the benefits of engagement and mitigated one of the possible costs, *i.e.*, the increased user browsing effort.

1.4.5 How to support going from offline to online?

In Chapter 7, we propose a recommender framework to accelerate the process of going from offline recommendation modeling to online field evaluation or the online studies of user-system interaction. Nowadays, more and more researchers have recognized that there is a gap between the offline recommendation algorithm advancement and the online improvement of user interactive experience in recommender systems. Some researchers have mainly focused on developing statistical models to estimate online results from a data perspective making certain distribution assumptions. I believe that we need to integrate both human and statistical theories to help translate dynamic interface and content manipulation to human behavioral or psychological experience change, and the best way to develop it is to accelerate the user-centric field evaluation of recommendation techniques. I designed, developed, and released Samantha [2] , an open-source framework to enable turn-key deployment of data-driven predictive recommender systems. The framework is extensible and has one centralized configuration file to control data processing pipeline, feature extraction, model building, updating and serving, and online field evaluation. Samantha has served as the framework for three projects of my research spanning across domains for both offline and online evaluation. It has served as the backend of MovieLens. Others also have started adopting it including other researchers in GroupLens lab [3] and some GitHub community users.

[2] https://github.com/grouplens/samantha
[3] https://grouplens.org

Chapter 2

Related Work

2.1 Classical Works in Recommender Systems

Collaborative filtering. Collaborative filtering based recommender systems have been proven successful and widely used in industries. The classical algorithms user-based or item-based [5] collaborative filtering algorithms compute similarities between users or items and make recommendations based on the nearest neighbors, *e.g.*, the most similar items to what users have rated high are recommended in item-based collaborative filtering [5]. The theory of collaborative filtering is however a social theory because the fundamental hypothesis behind it is that people who have similar ratings or tastes before will also have similar ratings or tastes in the future and this relationship is relatively stable [6]. The classical matrix factorization algorithm FunkSVD [7] is based on the theory as well and that's why it is also included under the name of collaborative filtering (although the border gets blurred after the development of machine learning community where the theories become learning theories). Although similar to item-item k-Nearest Neighbors (kNN), SLIM [8] takes another approach by directly postulating the sparse linear relationship of the predicted rating and user history ratings. SLIM starts to take the approach of learning and optimization, but maintain the flavor of some human theories, *e.g.*, global vs. local SLIM [9], which might correspond to human group perspectives on item similarities.

Context-aware and multi-criteria recommender systems. The theory of context-aware recommender systems [10] lies in that the suitability of a recommendation

depends on user context. Going beyond the two dimensions of user-item, context is incorporated as additional dimensions. In their work on Context-Aware Recommender Systems (CARS) [10], Adomavicius *et al.* examined how context can be defined and used in order to create more intelligent recommendations, such as using pre-filtering and post-filtering strategies with respect to contextual factors. Context can also be dynamic (vs. static), because system designers may find previously relevant context no longer useful, such as shopping companion. Correspondingly, the technique of tensor factorization [11] gets applied here to model the effects of context. At the same time, Adomavicius *et al.* [12] define the recommendation problem as a multi-criteria decision making problem (MCDM) and argue that the suitability of a recommendation for a particular user may depend on more than one utility-related aspect that the user takes into consideration when making the choice. These human theories are picked up by other machine learning techniques as well in recommender systems, *e.g.*, multi-objective optimizations [13], although the question that whether this multi-objective optimization approach can always help achieve an optimal solution across all criteria or whether it compromises some criteria while improving others is not yet fully addressed.

Dynamical Recommender Systems. Recommendations are typically dynamic. We use recommender dynamics here to broadly refer to the change in recommendations. There are many kinds of dynamics in dynamical recommender systems as reviewed by Rana *et al.* [14]. The most classic ones model user temporal preference drifting [15, 16, 17]. Koren [15] proposed to track user preferences on products along the whole time period in history data sets by explicitly postulating parameters regarding temporal effects and successfully incorporated this idea into two popular recommender techniques: a factorization model [18] and the item-item neighborhood model [5] to improve preference prediction accuracy. From the algorithm's perspective, recommendation changes are affected by how frequently the algorithm updates the models, *e.g.*, item similarities, user profiles etc. In the extreme case, online updating can be applied for each user interaction, *i.e.*, the technique of stochastic optimization [19]. Research on user-centric assessment of these online updating algorithms is rare in the field and my research takes the initiative to fill the gap.

2.2 Machine Learning Models

Generalized matrix factorization and decision tree based ensemble learning. Originally as a standard tool (*e.g.*, Singular Value Decomposition [20]) in linear algebra, matrix factorization techniques have been generalized to many different kinds of data going beyond co-occurrence or relational matrices in recommender systems borrowing the statistical theories of generalized linear models (*i.e.*, modeling the data through exponential family distributions) [21, 22]. For example, Gaussian distributions and L2 norm loss function in the corresponding optimization problem can be used for user rating data, while Bernoulli distributions and logistic loss function can be used for binary data of item displays and user actions on the item. Essentially, these models are treating user and item IDs as categorical input features in a generalized regression scenario. Other features, such as user demographics and item meta information in addition to user or item IDs, now can be freely incorporated [21], although this does not change the fundamental fact that these models are fitting the response distributions for whatever objectives that the system is designed to optimize for. Matrix factorization is where the border of human theories and machine learning theories gets blurred. Matrix factorization based on the collaborative filtering theory has evolved into low-dimensional embedding methods in machine learning based on learning theories. These methods can be widely applied in different types of datasets, but they have disadvantages as well because these models now has lost their foundation in human theories and it is not straightforward anymore to translate algorithm advancement into recommender system improvement, because the latter requires metric definition and user-centric research. In machine learning, quite different from embedding based models, decision trees such as CART [23] split the input space into a set of rectangular regions and model the distribution of response variables in the regions to make prediction. When using trees as components, powerful predictive models can be constructed through ensemble learning. Gradient Boosting Machine (GBM) proposed by Friedman [24] can boost any weak learner which has better than random performance to be a strong predictor. Particularly, a widely used model Gradient Boosted Decision Trees (GBDT) [24] is a GBM using trees as the weak learners. Random Forest [25], on the other hand, which represents another ensemble approach – bagging (v.s. boosting), achieves great performance

by averaging many decision trees mainly because of the variance reduction compared with each individual tree. GBDT has been used by industrial recommender systems, *e.g.*, Yahoo! News recommendation in [26].

Rating prediction for explicit feedback, action prediction for implicit feedback and learning to rank for relative preference. Historically, most recommender system research was based on explicit feedback data from users, *e.g.*, five-star ratings on movies, which includes the widely used MovieLens datasets [27] and Netflix competition and datasets [28]. In machine learning, it is typically formulated as rating prediction, or more traditionally the matrix completion problem [29]. Nowadays, researchers have switched from explicit ratings, *i.e.*, what users rate, to implicit behaviors or actions, *i.e.*, what users do. This is typically formulated as action prediction problems, *e.g.*, CTR estimation [30]. It requires generalizing the model to handle non-Gaussian data, *i.e.*, going from regression to generalized regression, *e.g.*, logistic regression type of models. Ranking metrics are widely used for top-N recommendations *e.g.*, Mean Average Precision (MAP) [3], normalized Discounted Cumulative Gain (nDCG) [3, 4] etc. and researchers also have been working on methods that directly optimize for the ranking metrics, which are the so-called *learning to rank* models [31]. From the user's perspective, there are two additional reasons that argue for going beyond simple regression and optimizing directly for ranking metrics. First, implicit feedback data has uncertainty and is typically relative. For example, if a user clicks something, we cannot assume this user likes it absolutely, but we might be able to assume it is better than other alternatives displayed. Second, some psychology research [32] has demonstrated that pairwise relative ordering is a better preference elicitation method, although better here specifically points to the accuracy of the elicited preference, not the amount of work users or systems need in order to have enough preference data. Among the various learning-to-rank algorithms, LambdaRank [33] generalizes machine learning models to be able to learn for most of ranking metrics by approximating the pseudo gradients in the non-smooth ranking world.

From supervised to reinforcement learning. A number of machine learning approaches are used in building recommender systems, including the following two common learning paradigms: supervised learning for instructive feedback [34, 8, 21] and reinforcement learning for evaluative feedback [35, 36] of which bandits learning is

a simpler version [37]. Models that learn from instructive feedback attempt to produce exactly the same response if given similar inputs. Models learning from evaluative feedback are aware that the feedback depends on how they act in the first place and hence treat not only producing a response but also collecting feedback as part of their decision [37]. A large corpus of machine-learning-based recommender system research currently focuses on the first paradigm, typically using offline data sets that have been collected previously. There is increasingly more research in recommender systems that go beyond traditional supervised machine learning acknowledging the limitation of the greedy fit to what the models see previously [35], *i.e.*, moving into the paradigm of reinforcement learning [36]. Within this paradigm, contextual bandits [35] represent an extended version of simple multi-armed bandit problems that have been used in recommender systems where arms can not only be individual decisions but also generalized feature descriptions of the decisions. Still, contextual bandit problems are a simpler version of the full reinforcement learning problem because the models assume an i.i.d. sampling process for the input. This essentially ignores the effects of the previous decisions on the environment, *e.g.*, the effects of recommendations on user retention. Recommenders based on Markov Decision Processes (MDP) [36] attempt to model the effects of recommendations on users and the systems, but the tractability of these types of model is still an issue [38]. Recently, Wu *et al.* [38] proposed to use recurrent neural networks for the state transition models avoiding estimating a full posterior distribution for the state variables turning into a partially deterministic approach and demonstrated its benefits. Generally, evaluating a reinforcement learning model in an offline setting is hard because the paradigm recognizes the biased property of historical data. Various methods to conduct offline evaluation through robust statistical importance sampling techniques have been proposed recently [39]. Based on this type of evaluation, researchers have been looking at possibilities of making recommendations based on their longer-term or even lifetime value of each user [40].

2.3 User Modeling

Visual attention models. To better understand and interpret user implicit feedback in recommender systems, a fundamental factor that can not be ignored is visual attention, *i.e.*, the question that whether users see the recommendations at all if they do not act upon them. From cognitive sciences, two main mechanisms guide the selection of human attention: top-down and bottom-up (or endogenous vs. exogenous) processes [41]. We can volitionally focus our attention according to top-down task demands. On the other hand, our attention may be drawn by bottom-up salient stimuli. This dichotomy of attention is still under debate because it involves the fundamental question of seeing attention as a cause, as an effect or as a combination of both [42], which has significant implication for interactive applications such as recommender systems. Specific to visual attention, I focus on reviewing overt attention [43] here, *i.e.*, gaze is directed to the attended location. Following Marr [44] and Itti's [45] seminal work, there has been plenty of research on modeling visual attention in a bottom-up approach, *i.e.*, saliency prediction [46, 47]. However, researchers have started to criticize the overemphasis on low-level saliency representation of visual input and develop new models of gaze allocation guided by top-down principles to account for complex natural vision [48]. Tatler *et al.* [49] showed that a model based solely on behavior biases and blind to current visual information can outperform a salience-based approach. Top-down task demands or regularities can be formalized with probability theories, especially Bayesian statistics. For example, Markov stochastic processes have been applied to model gaze transition behaviors, since it is intuitive to compare eye movements to random walkers. Ellis and Stark [50] developed a method to identify statistical dependencies in positions of eye fixations based on Markov matrices and found that there is statistical dependency among sequences of fixations independent of the physical placement of the points of interest. Further work built on this [51] modeled sequences of visual fixations as Markov processes and introduced a quantitative method to measure scanpath similarity based on character strings. Henderson *et al.* [52] demonstrated that it is possible to classify the task that a person is engaged in, *i.e.*, cognitive states from their eye movements by multivariate pattern classification specifically naive Bayes. Haji-Abolhassani *et al.* [53] modeled eye trajectories as a noisy generative process centered on the foci of attention

directed by cognitive processes using Hidden Markov Models. My research builds on this top-down approach and does not model the saliency of the displayed items on a page, but instead looks at how high-level information of item position presented in the interface directs and regulates user gaze behavior.

My research studies two hypotheses for user gaze behavior in a grid. In a grid-based layout, it is likely no longer valid to assume examining results from top to bottom one by one any more, since there are two potential directions (horizontal and vertical) that users can direct their attention. I examined two hypotheses regarding how users examine a grid: F-pattern [54] and center effect [55]. As pointed out by Tatler [55], observers have a tendency to fixate the center of the screen on computer monitors. They demonstrated the endurance of the central fixation bias irrespective of the distribution of image features, which implies that the center of a screen may be an optimal location for early information processing as learned by users. On contrary, because that a grid with rows and columns are different from an integral scene picture, the visual hierarchy might dominate the viewing behavior and users could exhibit a viewing pattern favoring the top and left sides [54], as suggested by the shape F going from top to bottom and left to right.

User browsing models. Granka *et al.* [56] pioneered the investigation of user behavior in WWW search through eye tracking analysis. They found that a higher rank in search results attracts more attention and users do tend to scan the result list from top to bottom. In information retrieval, machine learning algorithms are used to learn the relevance between search queries and web URLs from implicit clicking feedback [57]. Joachims *et al.* [58] examined the reliability of this kind of feedback generated from clickthrough data using eye tracking and explicit relevance judgement. They concluded that clicks are informative but biased, *i.e.*, the position bias because of search result presentation in a list layout. Following these findings, various user attention and browsing models are proposed to account for the bias in learning algorithms [59, 60, 61, 62] for information retrieval. Chapelle and Zhang [63] built and evaluated a dynamic Bayesian network model postulating explicitly examination or attention, action and satisfaction variables in addition to the observed clicking events. The temporal or dynamic aspect of the model lies in the assumption that users examine search results from top to bottom one by one, which is reasonable in a list layout interface and supported by previous

work [56]. Because of the cost of eye tracking, researchers have worked on approximating gaze in two main approaches: gaze-contingent displays [64] or restricted focus viewing and predicting gaze with mouse positions [65, 66]. As an example, Buscher *et al.* [67] compared segment-level display time of search results with eye tracking and found that although it is much coarser, it works as well as eye-tracking-based feedback for re-ranking and query expansion. Going beyond applications of information retrieval, Buscher *et al.* [68] worked on predicting gaze with web page location-based characteristics as input and generating a model that can be used to improve web page layout and design.

Eye tracking in recommender systems. Recommender system researchers have been using eye tracking in different ways. Since recommenders are considered an important decision support tool, Castagnos *et al.* [69] studied user decision making behaviors when purchasing products assisted by a recommender through eye tracking. They showed that users actively click and gaze at recommended products up to 40% of the time and consult the recommendation area more as they approach the end of the decision process. Another intuitive application based on eye tracking in recommenders is to infer preference or relevance from user gaze behavior. Xu *et al.* [70] proposed several algorithms to make recommendations relying on the attention time captured through commodity eye tracking as preference clues. Puolamaki *et al.* [71] combined eye movements and collaborative filtering [5] in proactive information retrieval tasks which is similar to a recommender and demonstrated its accuracy benefit in predicting whether a document is relevant. Recommender systems that rely on implicit feedback [72] could suffer from position bias as well, as demonstrated by Hofmann *et al.* [73] in simulated experiments. They examined this bias using different click models and showed how bias following these models would affect the outcome of recommender system offline evaluation based on implicit feedback data.

Human decision making theories and models. Decision making researchers have been developing different theories, which are generally divided into normative vs. behavioral models [74]. Normative theories are predictive and prescriptive, while behavioral theories focus on describing and also predicting human decision making behaviors. We focus on behavioral decision making here because we want to have a better understanding on what's actually happening in users' mind while they are looking at a page

of recommendations, not the optimal ways of browsing a set of recommendations. Two fundamental properties of human decision making have to been taken into account in order to be valid as a theory for explaining human decision making behaviors: determinism vs. probabilism (variability of preferences), statics versus dynamics (preference strength and deliberation time) [75]. Previous models failed to explain certain behavioral effects because of losing either one aspect of them, *e.g.*, expected utility model, random utility model, or dynamics of actions [75]. Decision field theory (DFT) [75, 76] on the other hand is a theory that takes into account both aspects and has been shown by previous research that can account for many prominent human behavioral effects. According to DFT, a decision is reached by the following deliberation process: as attention switches from one event to another over time, different affective values are probabilistically selected, these values are compared across actions to produce valences, and finally these valences are integrated into preference states for each action. This process continues until the preference for one action exceeds a threshold criterion, at which point in time the winner is chosen.

Other theories regarding human decision making have been proposed and studied, too. Well received ones are the Competing Accumulator Model [77] and ECHO model [78], in which the Competing Accumulator Model can account for all similar effects as DFT does (although through different mechanisms and further work is needed to discriminate between the two models). The ECHO model can account for the similarity and attraction effects but has not been shown to account for the compromise or loss aversion effects. What's interesting about the ECHO model is that it introduces a special node in the connectionist network, called the external driver, representing the goal to make a decision.

2.4 User-Centric Evaluation

From offline metrics to user perception and behavior. Recommender systems can be evaluated with offline metrics and online field experiments. Offline metrics sometimes make assumptions about online environments. One such important assumption is that the recommendation value decays going from the top to the bottom of a recommended item list. The nDCG [4] and weighted recall (or Breese's score [79]) evaluation

metrics, for example, assume exponential decay. My research re-examines this assumption because it may not be optimal to display all best recommendations at the same time. List-wise optimization have been shown to improve recommendations [80], which suggests that an optimal list may not be the same as a collection of individually optimal items. User-centric research in recommender systems has been increasingly more important. As pointed out by McNee *et al.* [81], offline recommendation accuracy on its own often is not a sufficient indicator of recommendation quality, and further work by Konstan *et al.* [82] elaborates the evolution of recommender system research from being concentrated purely on algorithms to research focused on the rich set of questions around the user experience with the recommender. Several frameworks have been proposed and widely used by researchers to evaluate and understand user experience in recommender systems. For example, Knijnenburg *et al.* [83] proposed a comprehensive framework taking into account both objective system measurements and subjective user perceptions to explain user experience. I apply this framework in my research. Particularly, they postulate six components and their causal relationships objective system aspects (OSA), subjective system aspects (SSA), user experience (EXP), user interactions or activities (INT), situational characteristics (SC) and personal characteristics (PC) – according to Theories of Reasoned Action (TRA) [84]. McNee *et al.* [85] proposed an analytical process model called Human Recommender Interaction that acts as a bridge between user information seeking tasks and recommendation algorithms to help with the design and structure of recommender systems. Pu *et al.* [86] proposed a user centric evaluation framework by employing state-of-the-art survey designs structured and derived based on theories of human behavioral intention and reasoned action. Particularly relevant to our research, the Unified Theory of Acceptance and Use of Technology (UTAUT) developed by Venkatesh *et al.* [87] postulates important social user factors that cause people to develop behavioral intention towards technologies and actual behavior of accepting or abandoning of the technologies. Work from Xiao *et al.* [88] is a direct application and further development of this theory in the domain of e-commerce recommender agents, *e.g.*, highlighting the importance of trust, perceived ease of use, and perceived usefulness in determining the user intention of future use of the recommender agents.

2.5 Recommender Toolkits and Machine Learning Libraries

There are a variety of softwares that have been built regarding recommendation or machine learning that can be used to build recommender systems. Some implements specific models or algorithms, *e.g.*, Apex SVDFeature [21], SLIMx [8] etc. which we categorize as specific tools. Some provide with implementations for multiple models or algorithms and especially provide their capabilities through programming language APIs, *e.g.*, Lenskit for collaborative filtering based algorithms [89], scikit-learn [90] and SparkML [91] for various kinds of machine learning models.

Researchers have developed general libraries to build models with complicated structures, *e.g.*, the recent popular platform TensorFlow [92]. Stan [93] has been developed earlier than TensorFlow although it is less scalable because it is oriented mainly towards offline data analysis. TensorFlow provides powerful modeling techniques but its support for hypothesis testing and other types of analysis is limited because of its focus on computational performance and production deployment.

Although most of the toolkits or libraries provide their capabilities through programming language APIs or sometimes console commands, Prediction.io [94] when integrated with SparkML is quite different because it is service-oriented. It has production serving and maintaining components and also integrates a variety of models and algorithms mostly through SparkML. This software is the most similar one to what I have built although there are also many differences as elaborated later in Chapter 7.

Chapter 3

Gaze Modeling in Grid-Based Interfaces

3.1 Introduction

Recommender systems research has experienced a transition from modeling user preferences based on explicit feedback [95, 5], *e.g.*, **what users are rating** to preference modeling based on implicit feedback [72, 96], *e.g.*, **what users are clicking**. Nowadays, it has been recognized that successful recommendations also need to take into account user perceptions of recommendation properties such as diversity and serendipity [3, 81], user short-term information needs, user context, and mood, *i.e.*, **what users are thinking**.

Understanding how users look at a recommender system's content will enable further improvements to how systems model and react to user needs. As users browse a recommender system, they systematically consider or skip over much of the displayed content. It seems likely that these eye gaze patterns contain a rich signal concerning these users' state of mind. Indeed, early studies have shown the potential for improving recommender systems by incorporating eye tracking data [70, 71].

The ability to incorporate eye tracking data into a recommender system enables a variety of potential improvements. For example, recommender systems currently do not know which items are looked at and ignored versus simply not looked at. But this is a critical distinction – if the user looks and does not act, that *inaction* provides a signal

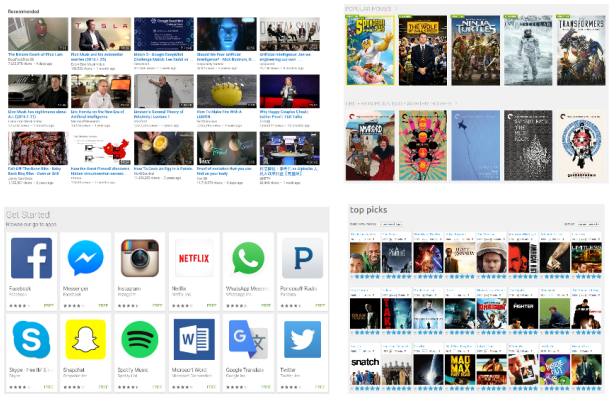

Figure 3.1: In this work, we predict user gaze in grid-based user interfaces. Above are four such layouts – YouTube (top-left), Hulu (top-right), Google Apps (bottom-left) and MovieLens (bottom-right).

that can be used to influence whether and when to display that item in the future, though interpreting whether such gazes represent interest or lack thereof may require context and further analysis.

Also, since recommender systems essentially provide decision support for users, having user gaze enables more nuanced studies on user high-level decision-making processes as demonstrated by researchers who study human decision theory through eye tracking [97].

The biggest challenge to incorporating user gaze data into recommender systems is a technical one: it requires eye tracking technology, which is generally not available outside of specialized labs. It is possible that future systems will make gaze detection a common feature available to system builders, due to the ubiquitous presence of high-resolution user-facing cameras. It is also possible that eye tracking data will never be commonly used due to the privacy concerns that such low-level tracking raises.

To address these challenges, in this work we show that it is possible to model and predict user gaze without requiring the deployment of ubiquitous eye tracking technology. We predict gaze by combining easily-collected user browsing data with eye tracking data from a small number of users. Our technique is therefore able to leverage a small amount of eye tracking data to infer gaze patterns for other users.

In this research, we model gaze in the context of a grid-based user interface layout, which has become one of the most common user interface layouts in recommender

systems. For example, this is the layout used in YouTube, Hulu, Google Apps, and MovieLens, as shown in Figure 3.1. We address the following three research questions:

- *RQ1: How accurately can we predict gaze on items in a grid-based interface?*
  - based on models trained with only **user browsing data**. (Specifically, item position in the interface, user dwell time on the page, and actions such as clicks, ratings and wishlistings on items.)
  - based on models trained with collected **eye tracking data** for a small number of users in addition to user browsing data.
- *RQ2: How is gaze distributed on different positions in a grid-based interface?*
- *RQ3: How does gaze prediction accuracy vary for different tasks or modes of usage?*

We make the following contributions in this work:

- We show that incorporating eye tracking data from a small number of users significantly boosts gaze prediction accuracy as compared with only using user browsing data, even though the eye-tracked users are different from testing users;
- We demonstrate that Hidden Markov Models (HMMs) can be applied in this setting and that they are better than baseline models in predicting fixation probability and capturing the interface regularity through Bayesian inference.

3.2 Related Work

Human attention theory. From cognitive sciences, two main mechanisms guide the selection of human attention: top-down and bottom-up (or endogenous vs. exogenous) processes [41]. We can volitionally focus our attention according to top-down task demands. On the other hand, our attention may be drawn by bottom-up salient stimuli. This dichotomy of attention is still under debate because it involves the fundamental question of seeing attention as a cause, as an effect or as a combination of both [98], which has significant implication for interactive applications such as recommender systems. Specific to visual attention, we focus on reviewing overt attention [99] here, *i.e.*,

gaze is directed to the attended location. Following Marr [100] and Itti's [45] seminal work, there has been plenty of research on modeling visual attention in a bottom-up approach, *i.e.*, saliency prediction [46, 47]. However, researchers have started to criticize the over-emphasis on low-level saliency representation of visual input and develop new models of gaze allocation guided by top-down principles to account for complex natural vision [48]. Tatler *et al.* [49] showed that a model based solely on behavior biases and blind to current visual information can outperform a salience-based approach. Top-down task demands or regularities can be formalized with probability theories, especially Bayesian statistics. For example, Markov stochastic processes have been applied to model gaze transition behaviors, since it is intuitive to compare eye movements to random walkers. Ellis and Stark [50] developed a method to identify statistical dependencies in positions of eye fixations based on Markov matrices and found that there is statistical dependency among sequences of fixations independent of the physical placement of the points of interest. Further work built on this [51] modeled sequences of visual fixations as Markov processes and introduced a quantitative method to measure scanpath similarity based on character strings. Henderson *et al.* [52] demonstrated that it is possible to classify the task that a person is engaged in, *i.e.*, cognitive states from their eye movements by multivariate pattern classification specifically naive Bayes. Haji-Abolhassani *et al.* [53] modeled eye trajectories as a noisy generative process centered on the foci of attention directed by cognitive processes using Hidden Markov Models. Our study builds on this top-down approach. We do not model the saliency of the displayed items on a page, but instead look at how high-level information of item position presented in the interface directs and regulates user gaze behavior.

Eye tracking and information retrieval. Joachims *et al.* [56] pioneered the investigation of user behavior in WWW search through eye tracking analysis. They found that a higher rank in search results attracts more attention and users do tend to scan the result list from top to bottom. In information retrieval, machine learning algorithms are used to learn the relevance between search queries and web URLs from implicit clicking feedback [57]. Joachims *et al.* [58] examined the reliability of this kind of feedback generated from clickthrough data using eye tracking and explicit relevance judgement. They concluded that clicks are informative but biased, *i.e.*, the **position bias** because of search result presentation in a list layout. Following these

findings, various user attention and browsing models are proposed to account for the bias in learning algorithms [59, 60, 61, 62] for information retrieval. Chapelle and Zhang [63] built and evaluated a dynamic Bayesian network model postulating explicitly examination or attention, action and satisfaction variables in addition to the observed clicking events. The temporal or dynamic aspect of the model lies in the assumption that users examine search results from top to bottom one by one, which is reasonable in a list layout interface and supported by previous work [56]. Because of the cost of eye tracking, researchers have worked on approximating gaze in two main approaches: gaze-contingent displays [64] or restricted focus viewing and predicting gaze with mouse positions [65, 66]. As an example, Buscher *et al.* [67] compared segment-level display time of search results with eye tracking and found that although it is much coarser, it works as well as eye-tracking-based feedback for re-ranking and query expansion. Going beyond applications of information retrieval, Buscher *et al.* [68] worked on predicting gaze with web page location-based characteristics as input and generating a model that can be used to improve web page layout and design.

Eye tracking and recommender systems. Recommender system researchers have been using eye tracking in different ways. Since recommenders are considered an important decision support tool, Castagnos *et al.* [69] studied user decision making behaviors when purchasing products assisted by a recommender through eye tracking. They showed that users actively click and gaze at recommended products up to 40% of the time and consult the recommendation area more as they approach the end of the decision process. Another intuitive application based on eye tracking in recommenders is to infer preference or relevance from user gaze behavior. Xu *et al.* [70] proposed several algorithms to make recommendations relying on the attention time captured through commodity eye tracking as preference clues. Puolamaki *et al.* [71] combined eye movements and collaborative filtering [5] in proactive information retrieval tasks which is similar to a recommender and demonstrated its accuracy benefit in predicting whether a document is relevant. Recommender systems that rely on implicit feedback [72] could suffer from position bias as well, as demonstrated by Hofmann *et al.* [73] in simulated experiments. They examined this bias using different click models and showed how bias following these models would affect the outcome of recommender system offline evaluation based on implicit feedback data.

Two hypotheses for user gaze behavior in a grid. In a grid-based layout, it is likely no longer valid to assume examining results from top to bottom one by one any more, since there are two potential directions (horizontal and vertical) that users can direct their attention. We have two hypotheses regarding how users examine a grid: **F-pattern** [54] and **center effect** [55]. As pointed out by Tatler [55], observers have a tendency to fixate the center of the screen on computer monitors. They demonstrated the endurance of the central fixation bias irrespective of the distribution of image features, which implies that the center of a screen may be an optimal location for early information processing as learned by users. On contrary, because that a grid with rows and columns are different from an integral scene picture, the visual hierarchy might dominate the viewing behavior and users could exhibit a viewing pattern favoring the top and left sides [54], as suggested by the shape F going from top to bottom and left to right.

3.3 Building Models for the Gaze Prediction Problem

We define a specific type of gaze prediction problem here – **Aggregated Fixation Prediction**. **Fixation** refers to the stationary period between saccades [99], in other words, the maintaining of visual gaze on the same location (we focus on predicting fixation here because in most human visual activities, we reply on fixations to take in visual information [99]). Consider a user browsing a page in a grid layout (examples shown in Figure 3.1), which has r rows and c columns and $r * c$ items in total. The problem is to predict **fixation probability**, *i.e.*, whether the user has fixated, and **fixation time**, *i.e.*, how long the user has fixated, on each of the $r * c$ items aggregating the entire browsing of the user on the page, given **item positions**, the user's **dwell time** on the page and the user's **actions** (*e.g.*, rating, clicking or wishlisting) on some of the items. Note that the unit of prediction is for each displayed item in one page view.

3.3.1 Building Linear Models

We start with linear models to predict users' fixation. With access to a group of users' fixation data, we can build supervised machine learning models to predict future fixations for this group of users and even for other users.

For predicting fixation probability, we build a mixed-effect logistic regression model (taking into account the correlation among positions by using a random intercept for each page view) using the following three groups of features:

- *Position* features: row index, ranging from 1 to r, and column index, ranging from 1 to c;
- *Dwell time*: log transformed seconds spent on a page;
- *1/minActionDist*: the inverse of the minimum distance to actions on a page. This feature encodes the insight that users are more likely to fixate on surrounding items when they interact with an item on a page. For example, if a user acts upon (*e.g.*, clicks) the second position, then it is likely that the user has fixated on the first or third position, which are small in terms of Euclidean distance calculated with row index and column index as the coordinates. Since there might be multiple actions, we take the minimum. We take the inverse of the minimum distance so that the coefficient of the feature is positive and the value of the feature equals zero with no action.

For predicting fixation time, we use the same set of features but a different model, a two-stage hurdle linear model [102]. This model handles zero inflation property in training data, *i.e.*, users have not fixated on many displayed items, by modeling each data point first through a logistic regression and then through a zero-truncated negative binomial regression.

To simplify result presentation, we refer to linear models without using *1/minActionDist* feature as **linearModel** and models using *1/minActionDist* feature as **linearModelActionDist** which are also summarized in Table 3.1.

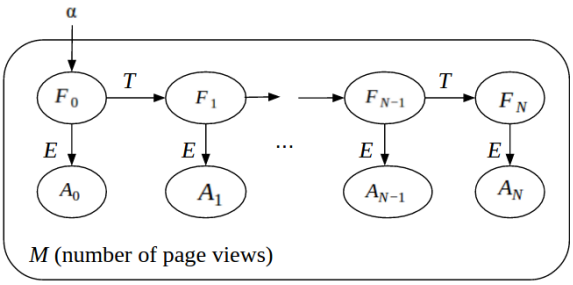

Figure 3.2: Graphical representation of a HMM in which F denotes fixation variable and A denotes action variable. α, T and E are parameters representing categorical conditional distributions defining the HMM. N is the length of the HMM sequence. For the gaze prediction problem, HMM gives inferred probability distributions of F when the values of A are observed.

3.3.2 Building Hidden Markov Models

We also use Hidden Markov Models, HMM for short, to predict users' fixation, as shown in Figure 3.2. For each page view, the dwell time is partitioned into small pieces of constant time intervals (the length of the interval is a parameter to tune). Each interval is associated with two variables: fixation F and action A. F takes $r * c$ possible values, representing the positions a user might fixate on the grid-based interface. We did not model no fixation (which could result from users' looking away or eye tracker's loss of gaze data. Time intervals with no fixation are removed from the observed F sequence) because it is less relevant for the prediction tasks. Given F, A can take $r * c + 1$ possible values, *i.e.*, one of the $r * c$ positions that a user acts upon plus the possibility of no action upon any position (we did not differentiate different types of actions). If multiple fixations are present in one time interval, we pick the one with longer fixation duration. If multiple actions are present (which rarely happens for small time intervals), we shift later actions to the following time intervals for which there is no action present or otherwise only use the first action. The parameters of this HMM are same for all users, *i.e.*, the HMM is not personalized:

- **initiation vector** α (length of $r * c$) represents the categorical probability distribution of the initial fixation.
- **transition matrix** T (size of $r * c$ by $r * c$) represents the categorical transition

probabilities from previous fixation to the next.

- **emission matrix** E (size of $r * c$ by $r * c + 1$) represents the categorical action probabilities given the current fixation position.

With fixation data from some users' page views (using eye tracking), we can estimate the above parameters by maximizing the likelihood of observing both the fixations and actions. We refer to this model as **eyetrackingHmm** (summarized in Table 3.1 as well).

Without fixation data, we can still estimate the above parameters with observed actions as follows. Firstly, we estimate E with one assumption – users usually do not act upon items they are not fixating on. Therefore, We set small probability values (specifically, $10e - 6$ in our algorithms) in E where positions of actions are different from positions of fixation. For the other case, we estimate the probability of acting upon f given that $F = f$ from frequency of actions on f in users' action logs. Secondly, we estimate α and T, by treating observed action as fixation, with the four algorithms named starting with *ub:* in Table 3.1.

Predicting based on HMM relies on the inferred *responsibility parameters*, *i.e.*, posterior distribution $P(F_i|A)$, denoted by a matrix R (N by $r*c$), in which N is the length of the HMM sequence. S_i defined in Equation 3.1 is computed for predicting fixation probability and L_i defined in Equation 3.2 is computed for predicting fixation time. The rationale behind these formulas lies in that $P(F_i|A)$ tells how much *responsibility* fixating on a position takes for the observed action sequence.

$$S_i = \sum_{j=0}^{N} R_{j,i}, \text{ for } i = 1 \text{ to } r * c \tag{3.1}$$

$$L_i = DwellTime * \frac{S_i}{\sum_{k=1}^{r*c} S_k}, \text{ for } i = 1 \text{ to } r * c \tag{3.2}$$

3.4 Methods

3.4.1 User Browsing Dataset in MovieLens

We focus on the interface called the **explore** page. It is a paginated three-rows-by-eight-columns grid-based layout presenting movie recommendations. Most of the explore page

views are completely filled with 24(= 3∗8) movie cards, as shown in Figure 3.1 (bottom-right).

We use a dataset with one month of user browsing data from November 2015. For our purposes, we track page dwell time and ratings, clicks and wishlistings on movie cards. When a user leaves the explore page by clicking a movie card and directly returns, we count it as a continuing page view of that explore page, rather than a fresh new page view, so the dwell time accumulates through interruptions such as movie detail page views. In total, this data set has 102,039 page views with associated dwell times.

3.4.2 Eye Tracking Protocol Design and Dataset

We collected 17 subjects' gaze data using Tobii T60 Eye Tracker (0.5 degree accuracy, 60 Hz data rate, 17" screen size, 1,280x1024 resolution, roughly 65 cm viewing distance). These subjects are university students, including twelve males and five females, aging from 18 to 25 and majoring across eight disciplines. Subjects reported that they had watched five or more movies in the past two months with two exceptions: one had watched two movies and another had watched three. They had never used MovieLens before. We set up an account for each subject and asked them to perform the five tasks listed below which takes around 30 minutes after the eye tracker calibration procedure.

- *Task 1: Browsing for fun (five minutes).* This is for the subjects to get to know MovieLens features and obtain natural gaze and browsing behavior.
- *Task 2: Rate 15 movies.* Subjects were instructed to rate based on their preference on movies in a five-star rating widget.
- *Task 3: Find 10 movies you'd like to watch given a three-month holiday.* Subjects were asked to add those movies into their wishlist using the wishlisting feature.
- *Task 4: Find 5 movies you'd like to recommend to your friends.*
- *Task 5: Find 5 movies you'd like to recommend to a 12 years' old child.*

We directly used the fixation records generated by Tobii Eye Tracker (see its manual for details of the algorithms to compute fixation from raw gaze[1]). Each record has

[1] `http://www.acuity-ets.com/downloads/Tobii%20Studio%203.3%20User%20Guide.pdf`

fixation duration and screen coordinates. To obtain fixations on the displayed movie cards, we recorded the movie card position coordinates, and tracked scrolling events as well to count for the position change. These positions are programmatically matched with the eye tracker's fixation records. Aggregating all 17 subjects, we collected 452 qualified page views (*i.e.*, views of explore page completely filled with 24 movie cards.). Since the unit of prediction is with respect to each movie card display, we have 10,848($= 24 * 452$) data points to use (Among them, we have 2304 for Task 1, 2760 for Task 2, 3960 for Task 3, 552 for Task 4 and 1008 for Task 5).

3.4.3 Evaluation

For each of the 17 subjects in the eye tracking study, we have their **true fixations** on each movie card in a page view, which is the **target variable** to predict. Prediction accuracy is measured with **AUC (Area Under the ROC)** for predicting fixation probability, *i.e.*, to classify a displayed movie in a page view into being fixated or not and **MAE (Mean Absolute Error)** for predicting fixation time. The unit of MAE is in seconds.

Depending on whether using true fixation data or not to train models, the evaluation has two scenarios. In the **training-with-fixation** scenario, both fixation and browsing data from the 17 subjects in lab settings are used. We randomly pick 4 (around 20%) subjects and use their data for testing, while reserving the other subjects for training. This procedure is conducted multiple times (around 100 runs; a different set of subjects are picked each time) to compute variance of the metrics. In the **training-without-fixation** scenario, as its name suggests, only user browsing data is used for training, including both the one month dataset and user browsing data from the 17 subjects in lab settings. In order to be able to compare the accuracy between these two scenarios, testing phase of this scenario uses the exact same fixation data as the previous scenario.

3.5 Results

RQ1: How accurately can we predict gaze on items in a grid-based interface? Figures 3.3 and 3.4 illustrate the accuracy of predicting fixation based on models trained with only user browsing data (the bottom five boxplots) and with eye tracking data (the

top three boxplots). First of all, we see that there is a significant accuracy boost resulting from training on eye tracking data even though the training and testing users are different. Specifically, AUC increases from 0.693 for *exactActionHmm* to 0.823 for *eyeTrackingHmm* ($p \approx 0$) and MAE decreases from 0.466 for *simpleActionStats* to 0.332 for *linearModelActionDist* ($p \approx 0$). This result demonstrates that gaze patterns are consistent even across different users, and that our models capture these patterns very well.

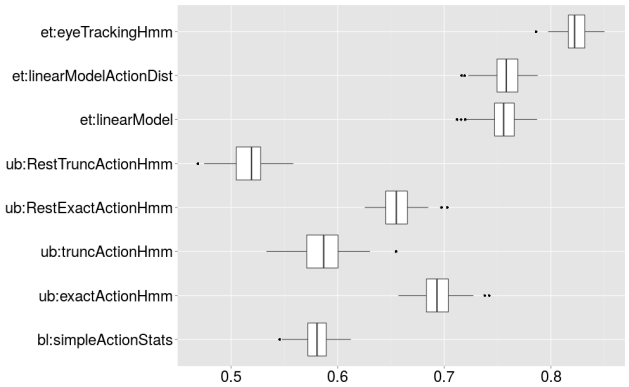

Figure 3.3: AUC boxplots for different models in predicting fixation probability. Higher scores are better. See Table 3.1 for descriptions of the models.

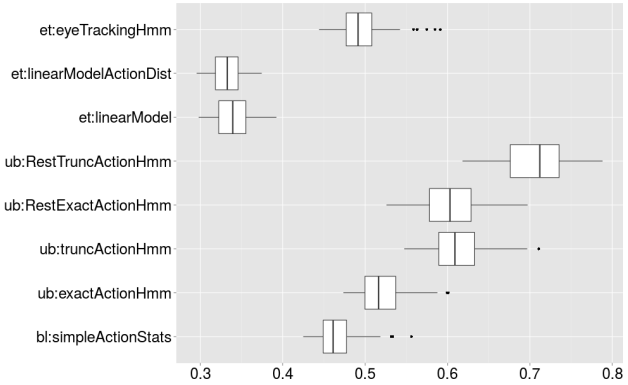

Figure 3.4: MAE boxplots for different models in predicting fixation time. Lower scores are better. See Table 3.1 for descriptions of the models.

In the *training-with-fixation* scenario, *eyetrackingHMM* performs significantly better than *linearModelActionDist* in predicting fixation probability ($AUC = 0.823$ vs. 0.757;

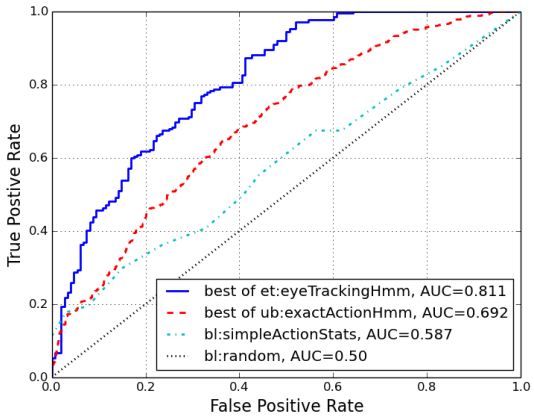

Figure 3.5: ROCs in classifying displayed movie cards into being fixated or not in a page view. It is from one run of the evaluation procedure.

$p \approx 0$). However, it performs worse in predicting fixation time ($MAE = 0.520$ vs. 0.332; $p \approx 0$). In the *training-without-fixation* scenario, *exactActionHmm* is much better than *simpleActionStats* ($AUC = 0.693$ vs. 0.580; $p \approx 0$) in predicting fixation probability (For an intuitive interpretation, Figure 3.5 shows the ROCs for one run of the evaluation procedure). But similarly, it has worse MAE (0.520 vs. 0.466; $p \approx 0$) in predicting fixation time. *RestExactActionHmm* and *RestTruncActionHmm* do not improve, which might be explained by overfitting to the action data set.

The above results show that HMM is more effective in capturing the interface regularity through Markov matrices and Bayesian inference in predicting binary-valued fixation vs. no-fixation, but is not very good at predicting real-valued fixation time. It might be explained by the choice of partition granularity in HMM, since we have to decide on a time interval. We are using one second for all HMMs after exploring multiple choices. It might illustrate a general difficulty of a generative modeling approach such as HMMs compared with a discriminative modeling approach such as hurdle linear models, in which fewer assumptions have to be made. Actually, hurdle linear models have better accuracy than ordinary linear regression, poisson or negative binomial regression and random forest with the same set of features. Note that MAEs less than a second do not imply that predicting fixation time is an easy task. It could possibly result from the small range of the ground truth values, especially with many zeros. Instead, we found

that it is hard to predict fixation time since with the best model we have, the prediction R^2(*coefficient of determination*) is 0.21. In other words, our model explains 21% of the variance in fixation time.

RQ2: How is gaze distributed on different positions in a grid-based interface? Figure 3.6 (drawn based on the mixed-effect logistic regression model; no significant interaction effects) illustrates user gaze behavior in a grid. It supports the *F-pattern* hypothesis, instead of *center* effect. Note that the fixation probability between either the first row and second row or the first column and second column is not significantly different. However, both the third row and third column have a significant drop ($p \approx 0$). Particularly, we omit the last column (index 7) because of data collection problem. The Tobbi eye tracker has relatively smaller screen size which leaves part of the movie card in the last column out of view. This however does not affect the conclusion for this research question. More interestingly, we found that for all positions dwell time is positively associated with fixation probability and when reaching 60 seconds, different positions on average have a very high probability (> 0.80) of being fixated.

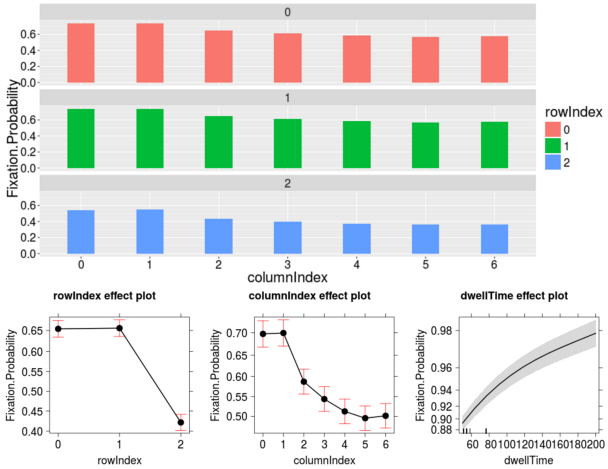

Figure 3.6: Fitted probabilities for different positions and the effects plot of *position* feature and *dwell time* in predicting whether a displayed movie is fixated using logistic regression. No significant interaction is found.

RQ3: How does gaze prediction accuracy vary for different tasks or modes of usage? From Figure 3.7, we see that Task 3 – *finding ten movies for self* – has the best accuracy in predicting fixation probability ($AUC = 0.842$, $p \approx 0$). Since more data is collected

for Task 3, it partially explains the accuracy advantage. Another possible explanation is that the process postulated by HMM particularly fits better to subjects' gaze behavior when engaging in this task. On contrary, as shown in Figure 3.8, the accuracy suffers most in predicting fixation time for the *finding-movies-for-children* task ($MAE = 0.340$, $p = 2.92e{-}08$). Subjects' gaze behavior shows substantial difference in this task from the video-recorded eye movements. Their fixations are shorter and more scattered, probably because subjects' searching strategy changed to coarser-level information scanning since most of the displayed items are not relevant anymore. Note that the better accuracy ($p \approx 0$) for Task 4 might just result from low variance in the data because we may not have enough data points for it. The general conclusion is that user gaze behavior is different in different usage modes and collecting and training on a specific task is better, especially for system designers who have knowledge about the main task that their users are engaged in.

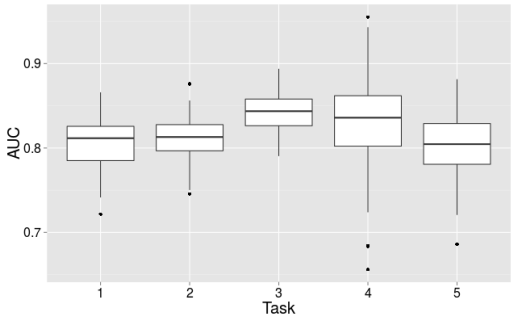

Figure 3.7: AUC boxplots of the best model *et:eyeTrackingHmm* in predicting fixation probability for the different tasks.

3.6 Discussion

Our gaze prediction techniques imply two direct practical applications in recommender systems. First, they could be used to improve recommendation freshness. We can predict which items the user has paid attention to repeatedly without action, and replace those items with new recommendations. Second, they could be used to remove potential position bias in preference modeling with implicit feedback [73]. We have tried to directly use the predicted fixation probability to weigh the click-through observations

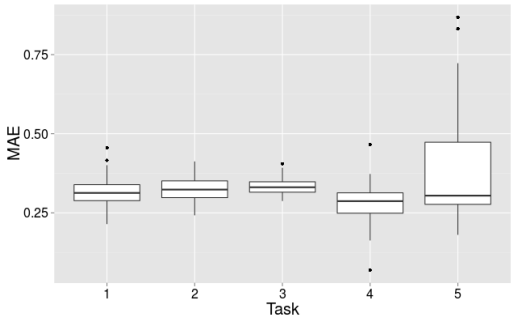

Figure 3.8: MAE boxplots of the best model *et:linearModelActionDist* in predicting fixation time for the different tasks.

in a matrix factorization model and achieved some accuracy improvement in predicting clicks under certain conditions. It does not always work because position bias is usually confounded with the typical relevance or preference of items shown at that position [63]. It is not straightforward to disentangle those confounding factors. A unified model on both gaze and preference may be necessary instead of simple weighting.

Table 3.1: Different models for the gaze prediction problem, in which *bl* denotes baseline, *ub* denotes training only on user browsing data (or *training without fixation*) and *et* denotes training on eye tracking data (or *training with fixation*) as well.

Model	**Model descriptions. Some depend on example action sequences a and b (nonzeros indicate the action position indices and 0 denotes no action)**
	Action sequence a: 0 0 2 0 4 0 3 0 0 0 0 Action sequence b: 2 0 3 4 0 0 0 2 0 0 0
bl:simpleActionStats (simple action statistics)	Based on simple action statistics, in which page dwell time is distributed among the $r*c$ positions proportionate to the frequency of actions on that position from user action logs. They are used in predicting both fixation probability and fixation time.
ub:exactActionHmm (exact action approximation)	a does not contribute on the estimation of α, because it starts with no action. b counts once for initiating from position 2. a does not contribute on the estimation of T because there is no action transition from a non-zero position to another non-zero position. b counts once for the transition from position 3 to position 4.
ub:truncActionHmm (truncated action approximation)	After removing zeros, a will count for initiating from position 2 and transition from 2 to 4 and from 4 to 3 even though that is not exactly what have happened.
ub:RestExactActionHmm	Re-estimating using Expectation Maximization (EM, [101]) algorithm with exactActionHmm as initial values.
ub:RestTruncActionHmm	Re-estimating using EM algorithm with truncActionHmm as initial values.
et:linearModel	logistic regression model for predicting fixation probability or hurdle linear model for predicting fixation time; *1/minActionDist* feature is not used
et:linearModelActionDist	same as above except that *1/minActionDist* feature is used
et:eyeTrackingHmm	α, T and E are estimated by Maximum Likelihood Estimation.

Chapter 4

Interpreting User Inaction Feedback

4.1 Introduction

Imagine one of your friends asked for a restaurant recommendation. You told her about a sushi restaurant nearby and she did not end up going there in the next week. What can we say about her preferences for restaurants and the reason why she did not go? And, if she comes for more recommendations after a week, would you recommend the same restaurant again to her? In the beginning you might ask her for a reason, but as time goes by, you might be able to learn that if she crinkled her nose and it was the second time you recommended the restaurant, she did not like the recommendation and you'd better stop recommending that one. On the other hand, if she looked upwards trying to remember and it was the first time you recommended, you might want to recommend it again since it is possible that she is interested but did not pay enough attention.

This type of scenarios happens similarly in online recommender systems, where users systematically browse items and decide to do something or not with the recommendations. In a typical online recommender interface, *e.g.*, the interface of Netflix, Amazon, or Youtube, grids or lists of recommendations are displayed to users once per page view. If users rate or consume (*e.g.*, watch or purchase) one of the items, the system learns from this explicit evaluative rating feedback or implicit behavioral action feedback to

make better future recommendations. For example, this feedback can be incorporated into (contextual) preference models to estimate what users prefer in general or in that specific context [18, 10]. However, among all that have been displayed, the majority of recommendations do not elicit any actions from users. We refer to this case as user *inaction*.

Intuitively, displaying recommendations triggered by user browsing affects user perception and experience with the system, and this should include both action and inaction. Just as actions provide feedback about user preferences, so do inactions, and this should be accounted for in the (contextual) preference models. For instance, a recommender that keeps recommending the same item again and again while ignoring user inaction feedback might not engage the user. On the other hand, a recommender that forgets what has been shown and keeps changing its recommendations based on user (in)action feedback could be confusing when the user is not able to retrieve a previously displayed and interesting recommendation, which the user has not yet had a chance to further explore.

Prior work in recommender systems has mainly focused on studying action rather than inaction [72], partially because inaction data is more ambiguous than action: inaction can (just like action) represent a deliberate decision, but can also result from not enough attention. Moreover, to understand the reasons for inaction, we need access to real users and their browsing activity data to better understand what reasons there are for inaction, and to build models to predict the type of inaction. With access to a live movie recommender system, we set out to answer the following research questions regarding user inaction by deploying a field survey about items not acted upon, analyzing and modeling the survey responses combined with a year of user browsing and interaction logs.

- **RQ1:** *What are the different categories of reasons for user inaction?*
- **RQ2:** *How do different categories of user inaction affect user future recommendation preferences for items not acted upon?*
- **RQ3:** *How well can we infer or classify the categories of user inaction from user log data?*

- **RQ4:** *Can we improve recommender systems utilizing a user inaction model based on our earlier findings?*

To answer RQ1, we examined several behavioral decision making theories (notably, Decision Field Theory by Busemeyer *et al.* [75] and the ECHO model by Guo *et al.* [78]) to come up with seven major categories of reasons for user inaction to drive our field survey design. In answering RQ2, we found that users demonstrated significantly different preferences regarding the future recommendation of inaction items that belong to different inaction categories. We then moved to RQ3, for which we investigated factors from user log data that might be predictive for inferring the category or class of the inaction recommendation. Among the significantly predictive factors, we observe some interesting but intuitive effects on the inaction class probabilities that we can infer. Finally, we have some evidence for RQ4, showing that taking account of the best inaction model's output, we can improve user action prediction substantially and potentially the timing of the recommendation, *e.g.*, delaying the recommendation to the next session according to the predicted probability of being interesting in future but not now for a previously displayed inaction recommendation.

Together our results show that user inaction is important to understand before making decisions on the future recommendation strategies of previously displayed items and the categories of user inaction can to some extend be inferred from user log data which can be further used to improve recommendations. In what follows we will first discuss related work, before discussing in detail the methods we used to answer our four research questions.

4.2 Related Work

Classical collaborative filtering algorithms are widely used in recommender systems, *e.g.*, matrix factorization techniques [18]. These techniques can be applied on user explicit rating feedback data, *e.g.*, using the factorization model to fit the values of the observed ratings and generate predictions for other unobserved user-item pairs. They can also be applied on user implicit feedback data, for instance, by treating the values of the observed acted-upon (*e.g.*, purchased, watched) items of a user as ones while other unobserved items as zeros [72]. These values are, however, associated with uncertainty

scores to represent that those feedbacks are not explicitly given by users and hence inherently uncertain.

Nowadays, more and more researchers and practitioners in recommender systems recognize that the interpretation of unobserved user-item pairs can be more complex because of the additional factor of whether and how items have been displayed to the user previously. Yang *et al.* [103] proposed the model of collaborative competitive filtering to reflect the fact that users make a decision of picking and acting on one item taking into account the competition of other context items displayed together. As one modeling approach, the work switches from using independent binomial model (through logistic loss) to using mutually exclusive multinomial model (through softmax loss). Lee *et al.* [104] studied how to estimate and utilize discounting functions of previous impressions (displays) to improve the conversion rate of recommendations, *i.e.*, based on how many times an item had been displayed and the last time the item was displayed to weight (down weight in the work because of the hypothesis that inaction tends to be negative feedback) the normal recommendation score of the item. This is similar to the cycling approach of manipulating top-N item lists proposed by us [105] which demonstrated an effect of increased user engagement, although we also observed negatively affected subjective user perception because of this manipulation.

The idea of utilizing binary (displayed and action or displayed but inaction) implicit feedback is more studied in information retrieval (IR) where one of the key tasks is to estimate the click-through rate (CTR) of a URL with respect to a search query to approximate or improve the search results quality. Joachims *et al.* [58] examined the reliability of this kind of feedback generated from click-through data using eye tracking and explicit relevance judgment. They concluded that clicks are informative but have the position bias, because of the search results' presentation in a list layout. Following these findings, various user attention and browsing models are proposed to account for the bias in learning algorithms [59, 60, 61, 62] for CTR estimation in IR. Recommender systems that rely on implicit feedback [72] could suffer from position bias as well, as demonstrated by Hofmann *et al.* [73] in simulated experiments. They examined this bias using different click models and showed how bias following these models would affect the outcome of recommender system offline evaluation based on implicit feedback data. The user browsing model could be substantially different for recommender systems compared

with information retrieval because modern recommender interfaces are typically grid-based, for which we [106] started collecting eye tracking data and modeling user visual attention.

Decision making researchers have been developing different theories for explaining decision making processes, which are generally divided into normative vs. behavioral models [74]. Normative theories theorize about optimal, often prescriptive and logical/rational approaches, while behavioral theories focus on describing actual human decision making behaviors often deviating from normative models. We take the perspective of behavioral decision making here because we want to have a better understanding of what actually happens in people's mind while they are looking at a page of recommendations, not the optimal ways of browsing a set of recommendations. Two fundamental properties of human decision making have to been taken into account in order to be valid as a theory for explaining human decision making behaviors: determinism vs. probabilism (variability of preferences), statics versus dynamics (preference strength and deliberation time) [75]. Decision field theory (DFT) [75, 76] is a theory that takes into account both aspects and has been shown to account for many prominent behavioral decision phenomena, such as for example context effects in which an additional third alternative influences the relative preference of two other alternatives (*e.g.*, the similarity effect, compromise effect, and attraction effect). The theory postulates a temporal comparative mental decision process when faced with several options and the accumulative preference of each option (called *valence*) dynamically changes while the decision maker is paying attention to different aspects of the options that are important for the decision. Once the valence reaches certain threshold, a decision will be made, *i.e.*, it is observed that one of the options is chosen. DFT provides a framework for us to formalize the attentional and competitive factors that might affect user behaviors seeing a page of recommendations. Other similar theories, such as the ECHO model [78], extend this work with a special contributing factor, called the external driver, representing the goal to make a decision. It suggests that context, tasks, and goals influence attentional processes in a dynamic way while users are browsing a page of recommendations. What users attend to (or not) reflects how they compare between options and part of the mental decision processes. In other words, action and inaction tells us more about the underlying preferences and might allow to improve recommendations when modeled

and taken into account.

Recommender systems can be evaluated with offline metrics and online experiments. Widely used offline metrics [3] include Precision, Recall, Mean Average Precision (MAP), Area Under the ROC Curve (AUC), Mean Average Error (MAE), and Root Mean Squared Error (RMSE). Unfortunately, these offline metrics have to make assumptions about online environments which mostly are not true, *e.g.*, assuming recommendation is a static ranking task to best recover or predict what users do in a held-out (future) part of the observed data sets. These metrics are limited because, *e.g.*, they cannot capture the dynamic interactive nature of how a recommender system is being perceived and used by people, like browsing page by page, going back and forth to compare or get more information to make decisions. As pointed out by McNee *et al.* [81], offline recommendation accuracy on its own often is not a sufficient indicator of recommendation quality, and further work by Knijnenburg *et al.* [83] and Pu *et al.* [86] proposed user-centric frameworks and evaluation metrics to answer a rich set of questions around user experience in recommender systems.

Inspired by the theories of behavioral decision making and following the user-centric approach, we set out to interpret user inaction in recommender systems from the perspective of understanding and improving user experience. Different from previous work utilizing item displays (or impressions) to improve recommender systems, our work first focuses on interpretation, *i.e.*, developing a specific model for why users do or do not interact with recommendations and further utilize this model to improve, so that when we do see improvement, we can better explain why it does and how it could behave in real systems from a user's perspective.

4.3 Data Collection

We collected user inaction survey data in the MovieLens system to answer our research questions. In order to collect survey data on user inaction, we summarized seven major categories of reasons for user inaction according to the postulated temporal decision making process from Decision Field Theory (DFT), [76], adapted to better fit the specific domain of the system. People normally do not watch most movies multiple times, *i.e.*, the re-consumption of movies could greatly affect whether users interact with a movie

recommendation or not. Some domains, however, see frequent re-consumption, *e.g.*, online grocery stores or music streaming services. From another perspective, whether a user has watched (or consumed) the movie before suggests that the user has a certain (highest especially right after the consumption) level of familiarity. A potentially important factor that contributes user inaction is lack of attention, as suggested by our eye tracking work [106] on grid-based interfaces and our survey in this work also supports this observation. When a number of recommendations are displayed in one page view, users probably will not pay attention to all of them especially when the item is displayed in a non-prominent position, *e.g.*, the right bottom corner in a grid-based interface. Even if a user does pay attention to an item, the user might prefer other alternatives that are displayed together or the user needs more information about the item to make a decision or just to find out whether it is actually less preferred compared with others. It could also be that the recommendation does not fit the user's movie taste, or that the user is looking for movies to watch with others so that additional constraints must be met. These scenarios reflect the effect of context described by the ECHO model [78].

We designed our field survey to be dynamically adaptive with multiple steps of questions for users to answer. Since our inaction interpretation derives from decision making processes, we excluded scenarios where users are not obviously making decisions based on the recommendations, *e.g.*, rating a previously watched movie or just browsing movie information. In addition, depending on previous answers to some of our survey questions, certain questions may not make sense. For example, if a user did not notice a recommendation, it would not be a valid follow-up question to ask why the user did not interact with it.

When a user goes to the *explore* page with 24 top picks and then transitions away from that page, we randomly picked one movie that was displayed on that page but not acted upon by the user (*action* includes rating, clicking, and adding into the wishlist, but excludes mouse hovering). If conditions to survey the user were satisfied (the user was asked fewer than four times before, and the time of last asking was more than one week ago), a survey was popped up to ask the user the following questions organized according to the flow shown in Figure 4.1. The order of the options to each question was randomly flipped (excluding the free text box) to avoid position bias. The short names

in the parentheses were not displayed but are included here for reference purposes. They also represent how multiple options are sometimes merged to make it easier for modeling, prediction and analysis (*i.e.*, NotNoticed in "notice", PastMonth/PastYear/YearsAgo in "when", Maybe in "future").

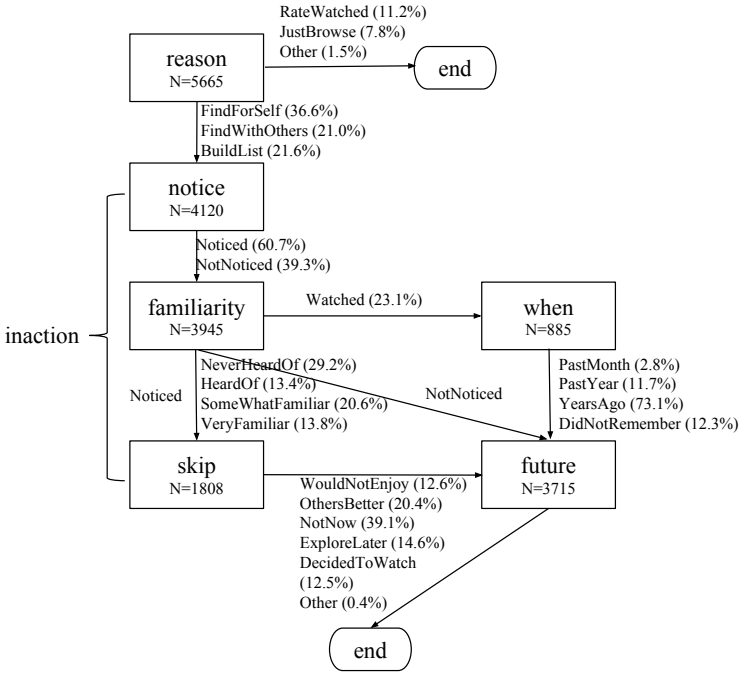

Figure 4.1: The flow of the inaction survey illustrating how the questions were asked. N is the number of responses to the corresponding question. The ratios are the proportions of options within those response.

- What was the primary reason you came to the MovieLens Top Picks today?(*reason*)
  - to find a movie to watch now or soon, probably by myself (FindForSelf)
  - to find a movie to watch now or soon, probably with someone else (FindWithOthers)
  - to build a list of movies to watch in the future (BuildList)
  - to browse movies without any specific plan to watch any of them in the near future (JustBrowse)

  - to find movies I've already seen to rate them (RateWatched)
  - other (free text) (FreeText)

- Did you happen to notice whether we displayed a movie recommendation XXX (1993) in the previous Top Picks page? (*notice*)
  - Yes, I noticed it (Noticed)
  - No, I didn't notice this movie being recommended (NotNoticed)
  - I don't think it was displayed, but I would have noticed it (NotNoticed)

- How familiar are you with this movie XXX(1933)? (*familiarity*)
  - Never heard of it (NeverHeardOf)
  - Heard of its name but don't know what it's about (HeardOf)
  - Somewhat familiar with it but have not watched it (SomeWhatFamiliar)
  - Very familiar with it but have not watched it (VeryFamiliar)
  - I've watched it (Watched)

- When did you watch this movie XXX(1933) last time? (*when*)
  - Past week (PastMonth)
  - Past month (PastMonth)
  - 1-6 months (PastYear)
  - 6-12 months (PastYear)
  - 1-3 years (YearsAgo)
  - $>$ 3 years ago (YearsAgo)
  - Don't remember (DidNotRemember)

- We noticed that you didn't interact with the card for movie XXXX (*i.e.*, you didn't wishlist, or click to see details)? Which best describes the reason why you didn't? (*skip*)
  - I already decided that I might watch it. (DecidedToWatch)

  - I was planning to click on this movie to explore it later; I just hadn't done so yet. (ExploreLater)
  - There were other movies recommended that seemed more interesting to me at the moment. (OthersBetter)
  - While I might be interested in this movie in the future, it isn't what I'm looking for right now. (NotNow)
  - I'm pretty sure I wouldn't enjoy this movie. (WouldNotEnjoy)
  - Other (free text) (FreeText)

- Should MovieLens continue recommending this movie to you in the future (until you rate it, of course)? (*future*)
  - Yes, definitely (Definitely)
  - Sometimes, but not always (Maybe)
  - Not now, but it would be nice to see it recommended again after some weeks or months (Maybe)
  - No, I'd rather get other recommendations (RatherNot)

From the survey question design, the seven possible categories of user *inaction* are labeled as *NotNoticed, WouldNotEnjoy, NotNow, OthersBetter, ExploreLater, Decided-ToWatch, Watched*, which integrates the questions of "notice", "familiarity" and "skip" (note that Watched here only includes those inaction items that are Noticed). We launched the survey on July 28, 2017 and collected user responses until March 21, 2018. 3,206 users gave 3,923 responses for which the user inaction category can be determined.

Along with the survey data, we also have access to user interaction logs in the system from Jan. 1, 2017 to March 21, 2018. These include 53M movie displays browsed by 25K users with information regarding how they were displayed (*e.g.*, position in the interface and how long the page dwell time was etc.), 1.6M ratings, 369K clicks, 167K wishlist additions, 2.8M hovers (hovering is only logged when the accumulative hovering time on the movie card is longer than one second within the page view) by those users.

4.4 Interpreting User Inaction

RQ1: What are the different categories of reasons for user inaction? Figure 4.1 illustrates the distribution of user responses to the survey questions. By focusing on the seven possible inaction categories here, we found that 38.6% of inaction recommendations were because of lacking attention (NotNoticed). 18.2% were because of lacking the right context (NotNow). 14.6% had already been consumed by the users (Watched), which were still recommended by the system because of lacking consumption records of the users. 9.5% were because of the effects of competition (OthersBetter). 5.8% did not match the user's taste (WouldNotEnjoy). 6.9% needed exploration later for more information to make a decision (ExploreLater). 5.8% had already reached the user's acceptance decision (DecidedToWatch) after that page view although it is an inaction recommendation. Lastly, outside of the options we provided for the "skip" question, we had 0.25% free-text responses. These numbers suggest that simply treating inaction as a signal of negative feedback (or simply ignoring the inaction feedback) could be problematic. Particularly, it points out that the effects of the two most important inaction factors – attention and context – need to be incorporated into the design of recommender models or the presentation of top-N recommendations.

4.5 Future Recommendation

In this section, we answer *"RQ2: How do different categories of user inaction affect user future recommendation preference?"* to demonstrate the significance of distinguishing different categories of user inaction. Figure 4.2 illustrates the distribution of future recommendation preferences for different user inaction categories. We conducted pairwise comparisons through six ordinal regression (specifically, mixed-effects cumulative link) models, assuming that "future" question has three ordinal levels: RatherNot, Maybe, Definitely (as mentioned previously, MaybeLater and Sometimes are merged into one level: Maybe), treating "inaction" as the fixed effect and user ID as the random intercept, varying the baseline condition of "inaction". To control false discovery, we employed Bonferroni correction [107] (effective significance p-value threshold is 0.0083; note the number of models built is six). The overall conclusion is that future recommendation preference can be statistically and substantially different for different inaction

categories. Specifically, there is a preferred order of future recommendation for the seven inaction categories. *From the least to the most preferred in future recommendation, the order of inaction categories is WouldNotEnjoy < Watched < NotNotice < NotNow or Others Better < ExploreLater or DecidedToWatch.* Note that movies with an inaction reason of DecidedToWatch are similarly preferred as ExploreLater, and users prefer being recommended inaction movies that they did not notice over ones that they noticed but had already watched.

As described in the survey design section, we also asked questions regarding "reason" (specific contexts), "familiarity", and "when" to watch. Figure 4.1 shows that the majority of user visits to the *explore* page top-picks are for finding movies to watch, although the context of finding the movie to watch may not be only for the user (*i.e.*, with others) or for immediate consumption.

We analyzed how "reason", "familiarity", "when" to watch might affect users' future recommendation preference by building similar ordinal regression models. We found that FindWithOthers has a significant negative effect on future recommendation preference compared with FindForSelf (coef.=-0.181, std.=0.090, p=0.044). It suggests that when users come to the recommender to find a movie to watch with others, the movies that they browse generally do not reflect their own preference and hence users prefer the system not to recommend these movies in future. For "familiarity", we found that Watched has a significant negative effect on future recommendation preference compared with not Watched yet ($p<0.001$) but we did not see significant differences among the cases from NeverHeardOf to VeryFamiliar. For different "when" options, we did not see significant differences either. However, we observe a trend that suggests users may be more likely to want to see a watched movie recommended in future when it was watched in the past year compared with the one watched either very recently or very far away in time.

4.6 Classifying User Inaction

The previous section shows that different categories of user inaction significantly affect the future recommendation preference of users. However, we are not able to gather data about user inaction on each of the displayed recommendations in the system. One

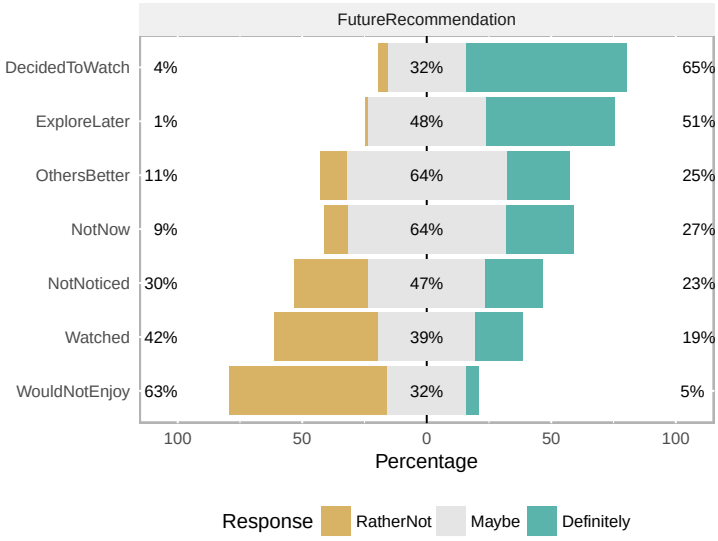

Figure 4.2: The distribution of future recommendation preference for different user inaction categories. The order of preference (significant after Bonferroni correction, p<0083) is *WouldNotEnjoy* < *Watched* < *NotNotice* < *NotNow or Others Better* < *ExploreLater or DecidedToWatch.*

possible alternative is to build classification models to predict, which is what our RQ3 is about: *How well can we infer or classify the categories of user inaction?* In order to answer this research question, we temporally split the survey data and system logs into two subsets. We used the subset of survey data and system logs before Feb. 1, 2018 (*i.e.*, Jan. 1, 2017 to Jan. 31, 2018) as the *training* data (around 90% of system logs, 80% of survey data). We used the remaining subset of survey data and system logs (*i.e.*, Feb. 1, 2018 to Mar. 21, 2018) as the *testing* or *evaluation* data.

The problem of inferring the category of user inaction can be formalized as a 7-class classification problem. Note that this is not predicting user inaction before a page of recommendations are displayed, but is inferring user inaction reason after observing how a page of recommendations are displayed and interacted with by the user. This classification model can be used for recommendation when we want to know whether we should re-recommend an item that has been previously displayed to the user before, which will be described in the last section. For now, we focus on answering RQ3.

We recognize that it might be helpful to first predict potential actions users might

Table 4.1: The confusion matrix of the inaction model (rows are the predicted classes and columns are the actual classes) and the accuracy in terms of AUC for each class (binary classification of one vs. others using the probabilistic output of the 7-class classification model).

Class	Watched	OthersBetter	DecidedToWatch	NotNoticed	NotNow	WouldNotEnjoy	ExploreLater	**AUC**
Watched	**96**	8	4	36	8	7	4	0.799
OthersBetter	0	**0**	0	0	0	0	0	0.720
DecidedToWatch	0	0	**0**	0	0	0	0	0.702
NotNoticed	59	29	21	**217**	80	22	27	0.696
NotNow	4	14	9	10	**16**	3	4	0.676
WouldNotEnjoy	0	0	0	0	0	**0**	0	0.621
ExploreLater	0	0	0	0	0	0	**0**	0.605

Table 4.2: Three possible ways to utilize the user inaction model in recommender systems. MF denotes Matrix Factorization and FM denotes Factorization Machine.

Goal	**Model**	**Metric**
Rating prediction (regression, L2-norm loss)	MF: user ID + item ID	MAE=0.912 RMSE=1.11
	FM: user ID + item ID + (7-class predicted probabilities)	MAE=0.950 RMSE=1.17
Action prediction (whether action or not when displayed; binary classification, logistic loss)	MF: user ID + item ID	AUC=0.774
	FM: user ID + item ID + (7-class predicted probabilities)	AUC=0.787
Recommendation timing	predicted NotNow probability vs. time taken for action	Pearson=0.0264 p<0.001***
	predicted NotNow probability vs. whether acted in a different session	AUC=0.562

make on recommendations before inferring user inaction. For instance, estimated probability of displaying (*predDisplay*) a movie by the system might signify the probability of ExploreLater. Estimated user preference (*predRating*) might signify the probability of WouldNotEnjoy. Estimated probability of rating a movie might (*predRate*) signify the probability of Watched. Estimated probability of clicking a movie (*predClick*) might signify the probability of ExploreLater. Lastly, estimated probability of adding into the wishlist (*predWishlist*) might signify the probability of DecidedToWatch. Therefore, we first built five models (referred to as *sub-models*) to generate these predictions through the classical matrix factorization technique [18] (latent factor dimension is 32). These action models (except rating value prediction which is a regression problem) only have positive items observed (*i.e.*, what movies were displayed, rated, clicked or added) for

which we need to sample negative items. For each item that was acted upon, we randomly sampled 2K items from the whole item space (≈45K) after which the classical matrix factorization technique can be applied (rating prediction uses L2-norm loss while action prediction uses binary logistic loss). The accuracy of these models is MAE=1.24, RMSE=1.51 for rating value prediction, Precision@1=0.512, 0.166, 0.018, 0.013 for predicting the probabilities of being displayed, rated, clicked, wishlisted respectively. We see that predicting rating, whether to display or rate are easier tasks than predicting whether to click or add into the wishlist in the system.

With the pre-built models, after systematically examining the factors that are potentially predictive, we summarized the following list of predictors.

- *item level*
  - popularity of the movie, *i.e.*, the number of ratings the movie has in the system
  - predicted rating, likeliness of displaying, rating, clicking and adding into the wishlist from the *sub-models*.
  - position where the item was displayed
- *page level*
  - user dwell time on the page
  - the ratio of *action* (used to refer to either clicking, rating or adding into the wishlist except hovering) and hovering on the 24 movies (*i.e.*, the number of acted movies divided by 24)
  - the min, max, median and mean of rating value, action and displaying probability predictions for the page of movies
  - the closest acted item's position in the grid, its Euclidean distance from the inaction movie (inversed), rating value and action predictions, if there is any action on the page
  - the min, max, median and mean of the similarities of the other movies displayed together with the inaction movie (these are cosine similarities computed on the movie latent factor representation from the rating value matrix factorization model)

- *session level*
  - the length of the session in seconds and how many movies were displayed before the page view
  - the ratio of hovering, action on the displayed movies before the page view
  - how many times the inaction movie has been shown in the session before the page view (this is further separated into two types of displays: displays on the *front* page and displays on the *explore* page).
- *user level*
  - the tenure of the user in the system in seconds and how many movies were displayed to the user before the page view
  - the ratio of hovering, action which includes either clicking, rating or adding into the wishlist except hovering on the displayed movies before the page view. Rating is further divided into lowRating(<4.0) vs. highRating(>=4.0).
  - how many times the inaction movie has been shown across sessions in the user history before the page view

The inaction model serves two purposes in this work: 1) understanding what predictors and how these predictors help infer the inaction categories and 2) achieving usable accuracy so that a recommendation algorithm can utilize its output to improve future predictions. Therefore, we employed two types of techniques respectively for these two purposes. First, for the purpose of interpretation, we employed multinomial regression to both test the significance of the predictors and their effects' signs and sizes. Second, to achieve the best classification accuracy, we used Gradient Boosted Decision Tree (GBDT, boosted ensemble of decision trees) model [24] and the popular implementation: xgboost [108]. This implementation supports sparsity regularization (similar to the technique of LASSO [109]) which enables automatic feature selection by tuning the regularization strength parameter α. After tuning, we found the best parameter set to be: $max\_depth = 1$ (the maximum depth of each tree), $num\_trees = 51$ (total number of boosted trees), $\alpha = 5.0$ (L1-norm based sparsity regularization), $\lambda = 1.0$ (L2-norm based regularization). The best model accuracy (7-class classification accuracy) is

48.5%, which is significantly better (8.6% accuracy boost) than the naive baseline of always predicting the majority class (p=0.0001 based on exact binomial test; the number of testing survey responses is 678; the majority class NotNoticed occupies 39.9%).

Table 4.1 shows the confusion matrix and the accuracy in terms of AUC for each class (binary classification of one vs. others using the output of the best 7-class classification model) of the inaction model. It shows that the model is struggling in differentiating other classes from the majority class NotNoticed. However, the probability scores predicted by the model still have certain capabilities to different each class from others as suggested by the AUC metric, specifically the model performs best in predicting for the classes of Watched, OthersBetter, DecidedToWatch but performs worst for the classes of ExploreLater, WouldNotEnjoy. Generally, inferring inaction categories is a hard task.

Because of the use of the sparsity regularization, GBDT model can provide non-zero-importance predictors after regularization. The effects of non-zero-importance predictors on inaction inference (from one multinomial regression model for interpretability) are listed in Table 4.3. Note that the coefficients in Table 4.3 are the log odd-ratio change of the corresponding category compared with NotNoticed. As two examples,

- The higher predicted probability of a user to rate a movie, the more likely that the user has watched the movie before. It suggests that predicting whether an item will be rated by a user in the system can approximate the user's familiarity on the item. (the cell in *predRate* & Watched)

- The higher the *predWishlist* is, the more likely the reason for inaction is DecidedToWatch. It suggests that predicting whether a movie will be added into the wishlist by a user in the system can approximate the likeliness of a user deciding to watch the movie, which is consistent with the design goal of the wishlist feature of the system.

As shown in Table 4.3, the predicted action probabilities can be useful signals in inferring the reasons of inaction for displayed recommendations. Although the possible actions that users can do in different systems vary, we see many interfaces in modern recommender systems such as Netflix, Youtube support similar actions as rating, clicking or adding into a wishlist. Therefore, these findings could potentially generalize across multiple platforms. However, future research is necessary to further validate this.

4.7 Improve Recommendation

In this section, we answer *RQ4: Can we improve recommender systems utilizing the user inaction model?* There are three possible ways of utilizing the inaction model to improve recommender systems.

- Preference estimation. Can we improve rating prediction accuracy by utilizing the inaction model output?
- Action prediction. Can we improve action prediction accuracy by utilizing the inaction model output?
- Recommendation timing. Can we do better in terms of when to recommend by utilizing the predicted probabilities of NotNow from the inaction model?

We use similar training and testing data sets as previous sections to answer these questions. However, we only took part of the system logs because it is expensive to extract predicted inaction category probabilities for all the 53M movie displays. To put these models into real systems requires amortizing the computational costs of running the additional models. For each of the 25K users, we take ten page views and their user interactions before Feb. 01, 2018 as the training set (176K ratings, 4.4M movie displays) and take one page view and their interactions on and after Feb. 01, 2018 as the testing set (7K ratings, 135K movie displays). Table 4.2 shows the results of testing the three possible ways of utilizing user inaction model output. The rationale of these approaches is that if we want to know whether or when we should recommend an item to a user, we first check whether we have displayed this item to the user before and generate predicted inaction probabilities as input to guide our decision. We used the technique of Factorization Machine [110] (FM) to incorporate the additional inputs of predicted inaction category probabilities. When an item was never displayed before, we use a default values of zeros as the additional input to FM. For both Matrix Factorization (MF) [18] and FM, we used 32 latent dimensions. To answer the recommendation timing question, we select inaction items that were later acted upon by users and analyze the Pearson correlation between the predicted NotNow probability of each inaction item and the time it takes for the user to act on it later. We also use this predicted NotNow

probability to predict whether the action was in a different session measured in terms of AUC.

As illustrated in Table 4.2, we did not see improvement for rating prediction in terms of MAE or RMSE, *i.e.*, estimating user preference, but saw substantial improvement in action prediction in terms AUC (predicting whether there will be any action on an item if it is recommended). It suggests that the inaction model can potentially improve recommender systems that targets maximizing user action (user engagement) in the system. We also observe that the predicted NowNow probability of the inaction model may help the system make the decision of delaying the recommendation of an inaction item to the next session or later in time.

4.8 Discussion

In online recommender systems, there are many possibilities to explain user inaction on recommendations. In this work, we summarized and collected data on seven major categories of them inspired by the psychology literature of behavioral decision making and found they significantly affect user future recommendation preferences on inaction items. In recommender systems literature, inaction or missing observations is usually treated as negative feedback [104, 72, 111]. Our research suggests that inaction is more complex than the assumption. For example, there is a high chance that the cases of ExploreLater and DecidedToWatch inaction are positive user feedback. We found attention plays a significant role in user inaction, which implies that new ways of presenting top recommendations might be necessary to better utilize user attention.

We designed and tested models to infer user inaction so that systems can avoid always asking users for inaction reasons. We achieved significantly better-than-random classification accuracy especially for certain inaction categories, *e.g.*, Watched, Others-Better or DecidedToWatch. We found interesting predictors that signify how we might better infer the inaction category probabilities, *e.g.*, predicted wishlist probability signifies a higher chance of DecidedToWatch and predicted being-rated probability signifies a higher chance of Watched. Generally, we found that user inaction inference is a hard task. However, with the advent of more accessible sensors like portable eye-tracking

equipments, we consider it promising further work to explore how these new measurements can help better infer user inaction.

We demonstrated that the user inaction classification model we built can improve action prediction tasks which can be used by recommender systems to maximize user action engagement (*i.e.*, recommending items that have the highest predicted action probabilities). We also showed that the predicted probability of NotNow from the inaction model could potentially improve recommendation timing, *e.g.*, delaying a previously displayed recommendation to the next session if the model predicts that this item can only be interesting to the user in future but not now.

Table 4.3: Coefficients (with standard errors) of predictors from a multinomial regression model predicting the category of user inaction. They represent the log odd-ratio change with respect to the baseline category NotNoticed. "closest" denotes the closest item that has an action if there is any action on the page. "row" or "col" denotes the row or column index of the position in the grid. "Sim" denotes similarity score. "numShow" denotes the total number of displays while "numFrontShow" denotes the total number of *front-page* displays. "[action]Ratio" denotes the ratio of items having the corresponding [action]. *closetInvDist* represents the inverse of the Euclidean distance between the acted-upon item (if there is any) and the inaction item. Significance levels: *p<0.05, **p<0.01, ***p<0.001.

Level	Predictor	WouldNotEnjoy	NotNow	OthersBetter	ExploreLater	DecidedToWatch	Watched
item	*predWishlist*	**10.912 (3.455)** **	1.249 (11.793)	-0.813 (3.121)	-0.052 (1.243)	**4.668 (1.751)** **	-7.124 (13.426)
	predDisplay	3.963 (2.785)	0.058 (7.334)	-3.441 (5.578)	**-5.332 (1.709)** **	-1.402 (1.421)	5.773 (7.508)
	predRating	-0.244 (0.282)	0.069 (0.190)	**0.699 (0.267)** **	0.160 (0.290)	-0.018 (0.324)	0.238 (0.224)
	predClick	0.372 (2.754)	1.974 (18.183)	3.065 (9.895)	-2.920 (2.545)	-0.945 (2.099)	-0.238 (18.831)
	predRate	5.197 (2.657)	-1.875 (6.307)	-5.322 (4.063)	**-5.501 (1.382)** ***	-1.602 (1.238)	**16.935 (7.122)** *
page	*row*	-0.004 (0.133)	**-0.209 (0.088)** *	-0.136 (0.110)	-0.139 (0.132)	**-0.320 (0.151)** *	**-0.243 (0.089)** **
	col	**-0.174 (0.044)** ***	**-0.154 (0.028)** ***	**-0.081 (0.037)** *	-0.077 (0.043)	-0.051 (0.048)	-0.047 (0.031)
	dwell	**0.180 (0.072)** *	0.075 (0.052)	0.099 (0.067)	**0.146 (0.074)** *	**0.217 (0.075)** **	**0.206 (0.055)** ***
	closestCol	**0.102 (0.052)** *	**0.100 (0.035)** **	0.051 (0.044)	0.000 (0.051)	0.064 (0.059)	-0.033 (0.037)
	closestInvDist	0.002 (0.227)	0.134 (0.148)	**0.395 (0.194)** *	**0.435 (0.220)** *	-0.035 (0.251)	**0.385 (0.156)** *
	minSim	-1.245 (1.117)	**-1.930 (0.702)** **	**-1.947 (0.881)** *	-1.002 (1.091)	-2.178 (1.234)	0.663 (0.775)
	medianSim	-4.272 (3.216)	-3.221 (1.976)	-4.939 (2.671)	**-7.648 (3.199)** *	-4.333 (3.410)	-4.025 (2.208)
	meanSim	4.425 (4.465)	**6.532 (2.692)** *	**9.196 (3.556)** **	**9.030 (4.494)** *	6.169 (4.824)	2.532 (3.140)
	maxSim	**-3.073 (1.212)** *	-1.425 (0.762)	-1.923 (0.983)	-1.980 (1.158)	-1.090 (1.299)	-1.002 (0.810)
	closestSim	**0.621 (0.308)** *	0.419 (0.214)	0.182 (0.261)	-0.148 (0.321)	0.247 (0.363)	0.015 (0.221)
	clickRatio	-1.645 (7.072)	**-12.134 (4.887)** *	-4.474 (5.853)	1.977 (6.448)	-6.812 (8.460)	**-24.583 (6.879)** ***
	ratingRatio	-0.661 (1.191)	0.418 (0.831)	0.420 (0.984)	0.984 (1.176)	-0.508 (1.387)	**-1.726 (0.800)** *
	lowRateRatio	**6.840 (2.256)** **	3.584 (1.852)	**4.497 (2.187)** *	1.620 (3.055)	**7.191 (3.011)** *	**4.603 (1.795)** *
	predRateMin	-0.065 (0.408)	-1.257 (0.767)	-0.533 (1.003)	**-1.321 (0.369)** ***	**-1.175 (0.241)** ***	**3.486 (0.795)** ***
	predRateMean	1.231 (1.495)	-1.045 (2.689)	-1.831 (2.741)	**-2.797 (1.072)** **	**-2.330 (1.017)** *	**7.225 (2.413)** **
	predRateMedian	0.714 (1.280)	-1.959 (2.379)	-1.800 (2.564)	**-2.668 (0.953)** **	**-2.436 (0.820)** **	**8.059 (2.235)** ***
	predRatingMin	0.041 (0.214)	-0.134 (0.122)	0.124 (0.165)	0.298 (0.230)	-0.155 (0.199)	0.463 (0.274)
	predRatingMean	-0.536 (2.146)	1.168 (1.302)	-2.417 (1.612)	-2.719 (2.034)	1.620 (2.338)	-1.041 (1.951)
	predRatingMedian	0.600 (1.788)	-0.064 (1.077)	**2.860 (1.407)** *	2.825 (1.726)	-0.138 (1.981)	0.335 (1.511)
	predRatingMax	-0.368 (0.570)	0.183 (0.317)	-0.159 (0.450)	-0.057 (0.556)	0.132 (0.551)	0.164 (0.439)
	predClickMin	-0.055 (0.039)	**-0.243 (0.100)** *	0.049 (0.118)	**0.160 (0.043)** ***	0.009 (0.034)	**0.274 (0.108)** *
	predClickMean	1.272 (1.049)	1.969 (2.030)	0.688 (1.841)	-1.051 (0.743)	0.056 (0.809)	0.377 (1.724)
	predClickMedian	0.681 (0.493)	-0.006 (1.038)	0.805 (0.894)	-0.575 (0.364)	-0.140 (0.408)	0.460 (0.912)
	predWishlistMin	**-0.249 (0.060)** ***	**0.267 (0.129)** *	0.006 (0.174)	0.022 (0.058)	**-0.298 (0.035)** ***	-0.081 (0.154)
	predWishlistMax	4.412 (11.852)	13.995 (7.845)	1.154 (12.143)	-5.502 (16.207)	-4.041 (18.608)	1.706 (9.528)
	predDisplayMin	-0.102 (0.096)	0.327 (0.218)	**-0.692 (0.251)** **	-0.066 (0.098)	-0.066 (0.101)	**1.049 (0.149)** ***
	predDisplayMean	-0.197 (1.200)	-0.450 (2.288)	-1.284 (2.826)	-1.554 (1.067)	-1.181 (0.839)	**5.458 (2.218)** *
	predDisplayMedian	-0.030 (0.744)	-0.145 (1.390)	-0.760 (1.675)	-1.065 (0.632)	**-1.519 (0.536)** **	**3.016 (1.235)** *
	predDisplayMax	-5.349 (7.038)	-6.996 (15.833)	-5.690 (18.309)	-5.104 (6.797)	-2.694 (4.402)	**36.050 (16.626)** *
session	*numFrontShow*	**0.818 (0.252)** **	**0.646 (0.168)** ***	**0.659 (0.209)** **	**0.832 (0.237)** ***	**0.750 (0.253)** **	**0.600 (0.191)** **
	lowRateRatio	**14.226 (6.922)** *	-6.013 (4.337)	-5.836 (5.772)	**-24.322 (5.887)** ***	-11.919 (7.467)	-2.251 (5.417)
user	*length*	0.002 (0.023)	0.022 (0.016)	0.001 (0.020)	-0.007 (0.023)	-0.022 (0.028)	**-0.047 (0.017)** **
	numShow	0.203 (0.106)	**0.299 (0.066)** ***	**0.253 (0.088)** **	**0.251 (0.100)** *	**0.567 (0.111)** ***	-0.079 (0.095)
	ratingRatio	**-17.097 (8.493)** *	11.615 (9.645)	2.015 (10.951)	7.201 (5.728)	2.984 (3.826)	2.160 (8.145)
	highRateRatio	16.444 (8.744)	-12.647 (9.837)	-1.533 (11.134)	-6.516 (6.038)	1.408 (4.318)	-0.257 (8.222)
	lowRateRatio	-0.556 (9.815)	-5.772 (9.532)	-0.386 (11.086)	10.145 (6.072)	2.154 (5.632)	-2.212 (8.952)
	wishlistRatio	-5.888 (12.539)	3.325 (6.203)	**14.435 (4.925)** **	**13.525 (5.520)** *	0.213 (11.142)	**14.693 (4.442)** ***

Chapter 5

Cycling and Serpentining of Top-N Lists

5.1 Introduction

Recommender systems typically are optimized to produce a top-N list reflective of the most-highly recommended items a user has not yet rated. However, there are many reasons to believe that this order may not be the best order to present items to users, either within or across sessions. First, top-N does not consider whether a recommendation has already been displayed to the user before, that is, whether it is fresh vs. potentially stale. Second, presenting the standard top-N list may create an experience where continued exploration results in a sense of finding ever-worse alternatives recommended. In this work, we explore two alternatives to the standard top-N approach designed to address these concerns. ***Cycling*** recommendations demotes recommended items after they have been viewed several times, while promoting fresher recommendations from the lower portions of the list. ***Serpentining*** displays a "zig-zag" order, in which the best recommendations (*i.e.*, the top recommendations from a rating prediction model) are spread across several pages, offering high-quality items on each page as a user continues to explore. Cycling may happen within the same visit or across multiple visits, which we call ***intra-session*** or ***inter-session*** cycling. Intra-session cycling creates a more immediate and noticeable change but may cause confusion because potentially interesting recommendations may disappear when a user goes back to the previous page.

Inter-session cycling is less likely to have this problem but may not be noticeable because users have forgotten what they saw previously.

The high-level research question in this work is whether *cycling* and *serpentining* – as two perspectives of re-examining top-N list – improve user experience. However, we are not trying to optimize a particular user experience. We recognize that different experiences may require different approaches. A situation where a site recommends a single item cannot benefit from *serpentining*. A user who treats the top-N list as a "to-do" list, taking the top item each time, would not be served well by *cycling*. Rather, we want to see how these manipulations relate to user experience in the hopes of guiding designers in adopting them, or offering them to users. Similarly to the finding from Ziegler's work [112] that users are willing to accept a certain loss of accuracy in order to have more diverse recommendations, we expect that the perceived accuracy of recommendations may get reduced because of the manipulation; however, we test whether the accuracy reduction may be preferred in exchange for the exposure to a broader and "fresher" set of items.

With this as the goal of our research, we look at multiple metrics and several dimensions of user experience. We recognize that users also have different goals, including those who want to explore deeply and ones who simply want to find an item quickly. For this reason, we look at (a) a variety of user activities, including *engagement measures* (levels of usage) and *success measures* (numbers of items selected) as well as (b) a variety of self-reported reactions, including assessments of *quality*, *freshness*, *usefulness*, etc. We follow the framework proposed by Knijnenburg *et al.* [83] for user-centric evaluation of recommender systems. Four components of the framework are involved: **OSA** (Objective System Aspects, *e.g.*, recommender manipulations), **SSA** (Subjective System Aspects, *i.e.*, user perceptions on different aspects of the recommender), **EXP** (experience, *e.g.*, the overall perceived usefulness or satisfaction), and **INT** (interaction, *i.e.*, user activities or behavioral data in the recommender). It leads to our research questions *RQ1-RQ3* listed below. We combine SSA, EXP, and INT measurements to better understand user experience. As pointed out by Velsen *et al.* [113], interpretation of user behavioral data is often troublesome, and they suggest *triangulating* objective behavior data with subjective experience data (which is collected through surveys in our experiment). For example, increased page views could be representative of better

(more) user engagement, but it could also mean that users are forced to browse more in order to get useful recommendations. We are concerned that asking survey questions may have an effect on users' activities as well. Therefore, we also design another variation in addition to the above two manipulations: ***delayed*** asking vs. ***non-delayed*** asking. For users in the delayed asking condition, we only ask them survey questions after they joined our experiment for certain period of time (one month here) so that we can measure user activities in a setting that is closest to the production environment of an online recommender system for a while (which usually does not have surveys).

- **RQ1:** *How do cycling and serpentining recommendations affect user activities? (OSA ⇒ INT)*
- **RQ2:** *How do cycling and serpentining affect user perceptions and their overall experience with the recommenders? (OSA ⇒ SSA and EXP)*
- **RQ3:** *How does each user perception aspect contribute to user-perceived usefulness and overall satisfaction? (SSA ⇒ EXP)*

5.2 Related Work

The cycling approach proposed above creates a distinct type of ***presentation-controlled*** dynamic, as it controls the presentation of a recommended item and cycles it out when it has certain exposure. We use recommender ***dynamic*** here to broadly refer to the change in recommendations. There are many kinds of dynamics in dynamical recommender systems [14]. The most classic ones model users' temporal preference drifting [15, 16, 17]. Rana and Jain [14] classified the dynamics of recommender systems into six categories: temporal changes, online processing, context, novelty, serendipity, and diversity. We review dynamics in recommender systems from a different perspective here. From the literature, dynamics can be achieved with two approaches: *model-based* and *algorithm-based*. Model-based approaches include context-aware recommenders [10, 114] and systems explicitly modeling user preference change [15, 17]. For example, in their work on Context-Aware Recommender Systems (CARS), Adomavicius *et al.* [10] examined how context can be defined and used in order to create more intelligent recommendations, such as using *pre-filtering* and *post-filtering* strategies with respect to

contextual factors. In their classification, contexts can also be dynamic (vs. static), because designers may find previously relevant contexts no longer useful, such as shopping companion. Koren [15] proposed to track user preferences on products along the whole time period in history data sets by explicitly postulating parameters regarding temporal effects and successfully incorporated this idea into two popular recommender techniques: a factorization model [18] and an item-item neighborhood model [5] to improve preference prediction accuracy.

The second approach to achieve dynamics is through *algorithms*, *i.e.*, how to find the optimal solution for the specified model and how frequently to update the model as new data set becomes available. As an example, matrix factorization [18] is a popular technique for recommendation, in which user preferences are modeled with latent factors and learned from user-item interaction matrix. Recognizing that factorizing those interaction data matrices in batch has significant delay compared with the time of receiving feedback from users, online learning or incremental techniques have been proposed and tested for real-time model updating [115, 116]. Most of machine learning based approaches to recommender systems have the incremental processing capability. For example, *learning-to-rank* [33] techniques directly learn the relative ranking of items to a specific user from data, whose dynamics critically depend on the updating latency of the ranking models, *i.e.*, how quickly new available information is fed into the algorithm. Many recommender systems have both *model* and *algorithm* based dynamic perspectives, such as Markov Decision Processes (MDP) based recommenders [36] and contextual bandits [117] in computational advertising. In these techniques, recommendation problem is modeled as a dynamic decision making policy for an agent, and algorithms are designed to search the optimal solutions based on partially available and incrementally gained information such as "like" or "dislike" feedback from users.

However, there is a need for more systematic study of recommender models' and algorithms' *effects on user perceptions and experience*. Change in recommendations is a good thing when users perceive more freshness and less boredom, but also could be confusing when changes are highly unexpected or overly dramatic. In other words, several psychological factors (that may not be directly observable) may be involved in user's decision making and, therefore, a systematic user-centered approach is needed to evaluate their potential involvement. Users' exposure to recommendations can also

be studied by analyzing user actions, following the approaches and ideas from the science of *persuasion* and *marketing*. As Tellis's [118] work showed, advertising exposure has a nonlinear effect, in other words, repetitive exposure is necessary but has diminishing gain. Their and others' results [119] suggested that two to three ad exposures might be optimum. As discussed by Petty and Cacioppo [120] and also suggested by their results, repeating a persuasive communication tends to first increase and then decrease agreement. They proposed a two-stage attitude modification process: repetition enhances a person's ability to process a message in the first stage, and tedium and reactance are elicited by excessive exposures in the second stage. Similarly, this two-stage process might also apply in recommendations. Although CARS [10] adapt recommendations based on users' contextual state, *i.e.*, based on time, mood, or companion(s), there might be contexts that are hard to measure and very sparse data about them is available for each individual user. Therefore, repetitive recommendations may increase the chances that users process the recommendations in relevant contexts. In addition, we study user-perceived boredom and freshness associated with the dynamics through surveys. There is not much research on changing recommendations based on users' past exposure to recommended items. One thread of related research is CTR (Click-Through Rate) estimation in information retrieval [57], where documents with many exposures but no positive feedback from users are downgraded because their estimated CTRs become lower. Recommender systems can also utilize indirect feedback, such as clicks, which would be treated as an implicit preference signal [72]. In other words, when focusing on implicit users' feedback in response to displayed recommendations, a recommender can be designed to achieve similar dynamics. We do not use CTR as the primary evaluation approach in our work, because the system studied is not targeted at generating click-throughs, but rather at helping users have better experience with in exploring and finding movies (as measured in a much more holistic, comprehensive manner). Moreover, in our system users can see and rate movies without clicking through to detail pages, so informativeness of clicks as a primary evaluation measure may be limited. However, we do keep track of clicks as one of several indicators of user activities and engagement with the system.

Recommender systems can be evaluated with offline metrics and online field experiments. Offline metrics sometimes make assumptions about online environments. One

such important assumption is that the recommendation value decays going from the top to the bottom of a recommended item list. The *nDCG* [4] and *weighted recall* (or *Breese's score* [79]) evaluation metrics, for example, assume exponential decay. We propose to test this assumption, because it may not be optimal to display all best recommendations at the same time. *List-wise optimization* have been shown to improve recommendations [80], which suggests that an optimal list may not be the same as a collection of individually optimal items. Also, it has been shown that, in addition to accuracy, many other properties of a recommender are important aspects of user satisfaction [121, 81, 85, 86, 112], such as diversity, novelty, etc. Pu *et al.* [86] proposed a user-centric evaluation framework for recommender systems with state-of-art survey designs [122]. Knijnenburg *et al.* [83] proposed a comprehensive framework taking into account both objective system measurements and subjective user perceptions to explain user experience. We directly apply this framework in evaluating our manipulations. Particularly, they postulate six components and their causal relationships objective system aspects (OSA), subjective system aspects (SSA), user experience (EXP), user interactions or activities (INT), situational characteristics (SC) and personal characteristics (PC) – according to Theories of Reasoned Action (TRA) [84]. We use and model the former four components through methods of recording and analyzing user activities and survey responses here. This framework relies strongly on asking users their subjective experience through survey questions. In many examples of this type of studies, users typically interact one time with a system and then evaluate its performance. However, in our current study users can interact with a system over time, *i.e.*, over several sessions. Because of this, we vary the moment of presenting the survey questions, to see if querying the user experience might affect how users interact with the system.

5.3 Experiment Design

To answer our research questions, we conduct a field experiment in MovieLens (41,125 movies as of July 2016). We include users who have at least 15 ratings to make sure that we are testing on users who have a reasonable level of engagement with the system, since most of the active users have more than 15 ratings (as shown in the Results section). We also limit the study to include only users who have at least one session of usage in

the system, excluding the current session for reasons explained below.

We invite qualified users to join the experiment through a link displayed on the home page: "Would you like to experience a new movie recommender named Spirit in MovieLens?" (Spirit is the recommender name we use for all conditions in this study). After clicking the link, users see an informed consent page, which briefly introduces the experiment including information about potential survey requests. If they consent, they are randomly assigned into one of the experimental conditions. Users can opt out of using the experimental recommender at any time by clicking a link at the top right corner which says "Stop Using the Spirit". Note that *opt-out* here specifically means stopping using the experimental recommender, not completely dropping out from the experiment. Users are informed that they can contact us through MovieLens to remove their data from the system if they wish to withdraw entirely. This experiment was approved by our institution's Institutional Review Board.

We employ a *between-subjects* 3x2x2 factorial design. The first design factor is cycling, which takes three levels – *no cycling*, *inter-session cycling*, and *intra-session cycling* – as mentioned in the previous section. What we want to accomplish through *cycling* is to control the amount of exposure a recommended item can have on a user, favoring those items that are least exposed but only after they have been presented certain number of times. This is achieved by re-ranking top-N items first based on the number of previous presentations and then based on the predicted preference from state-of-art algorithms. We use a *presentation* to specifically refer to a movie card display in the grid layout of MovieLens. Instead of directly using the number of presentations, we calculate how many times a movie has been presented to a user, divided by *three* (rounded to the smaller integer), which we call *presentation score*, based on a history of presentation data tracing backwards to one session before a user joins the experiment (this is enabled by our inclusion criterion of the participants, *i.e.*, users who have at least one session before joining). The implication of this is that an item will first be exposed three times before the algorithm starts to downgrade the item's rank in the new list. The predicted preference (*i.e.*, rating) of an item comes from the popular *item-item collaborative filtering algorithm* [5] built on the historical data of user ratings on items in the MovieLens. The top-N list is first sorted by the presentation score ascendingly and then sorted by the predicted ratings descendingly to get the new top-N

list. Further, as mentioned before, two types of *cycling* *intra-session* and *inter-session* are designed to have different dynamical extent. For the *intra-session* type, we cycle the top-N recommendations once every time users go (or go back) to the home page even when it is within the same session. For inter-session cycling, we only cycle once when users come to the home page in a new session. We take 240 items as the (top-)N here. It spans 10 pages of movie cards (with each page displaying 24 movies) beyond which there is no manipulation on the recommendation list.

Another design factor, *serpentining*, takes two levels: *true* and *false*. What we want to accomplish here is to have a new list where users can see best items spread in multiple pages. When serpentining=true, we re-organize the top-N list based on the original rankings of the recommendations (*i.e.*, 1 through N). Specifically, we pick movies intermittently with a constant ranking interval M (= 4). This is achieved by first reshaping the $N - by - 1$ list into a $M - by - N/M$ (*i.e.*, 4-by-60 here) matrix in a column first order as illustrated in Table 5.1. Table 5.1 also gives the page index when users are requesting the *k-th* (k=1 to 10) page of their recommendations. Notice that the 9th and 10th pages span two rows because each row has only 12 movies left. The algorithm can be summarized as *Page-level Column First* and *Item-level Row First* (PCF-IRF) serpentining.

Table 5.1: Page-level Column First and Item-level Row First serpentining (PCF-IRF) algorithm illustrated with (top-)N=240, M (number of rows)=4. 1st, 2nd, kth are the page indices. M controls how scattered the new top-N list is in the original rankings and also how much change an item's ranking can have after cycling).

1st	1	5	...	93	5th	97	101	...	189	9th	193	197	...	237
2nd	2	6	...	94	6th	98	102	...	190	9th	194	198	...	238
3rd	3	7	...	95	7th	99	103	...	191	10th	195	199	...	239
4th	4	8	...	96	8th	100	104	...	192	10th	196	200	...	240

To better explain the two manipulations, Figure 5.1 and 5.2 illustrates how they work with simple examples. In Figure 5.1, there are four items in the top-N (N=4) list. Originally, they are ordered according to the predicted scores from the best to the worst. With the *cycling* manipulation, when we observe that the first two items have been displayed for three times, we demote the two items to the bottom of the list and hence indirectly promote other two items to the top so that users can see less exposed items. In Figure 5.2, there are eight items in the top-N (N=8) list and we can only

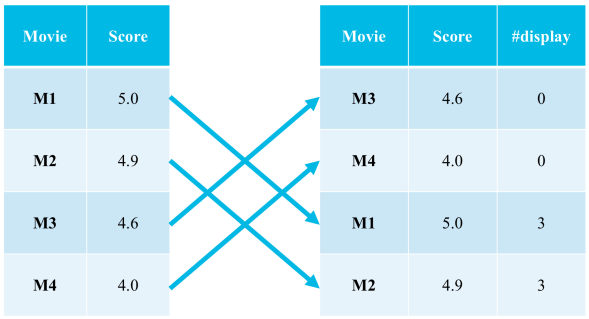

Figure 5.1: Illustration of how *cycling* works with an example.

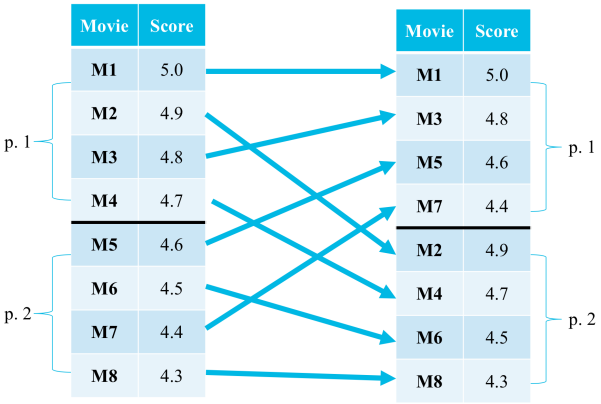

Figure 5.2: Illustration of how *serpentining* works with an example.

display four items in one page. Instead of displaying the items from the best to the worst according to the predicted scores, with *serpentining* manipulation, we fill out the positions of the pages by picking items with a fixed interval from the original list (here the interval is two, *i.e.*, picking one and skipping one) so that the best items are spread across the two pages. As we see, even when a user go to the second page, the user can still see the second best recommended item.

In the case where both *serpentining* and *cycling* algorithms are enabled (*i.e.*, the interaction between the two), we first apply the serpentining algorithm, then apply a cycling algorithm to the results. However, we only allow the cycling algorithm to affect ordering within columns as shown in Table 5.1. The goal of this design is to control the freedom of an item's new presentation position after cycling. For example, items in the second column of Table 5.1 with rankings 1 through 4 can exchange presentation

positions through cycling but not with other columns. It means the very best item may only go to the first of the 2nd, 3rd or 4th page. *Serpentining=false* actually is a special case where M=N, in which an item can be anywhere between 1 through N and may have dramatic position change, such as going from the 1st page to the 10th page. The third design factor ***asking*** takes two levels: *delayed asking* and *non-delayed asking*. For *non-delayed* asking condition, we survey users as soon as they have enough interactions (see the measurements for more details) with the experimental recommender. For *delayed* asking condition, we only survey users after they have joined the experiment for at least one month and also have enough interactions.

5.3.1 Measurements

Based on Knijnenburg *et al.* [83], we measure users' interactions with the system (INT), user perceived subjective system aspects (SSA), and user experience with the recommender (EXP). For INT, the following list of metrics are computed in a fixed period (*half a month* here) of time for *each user*.

- ***optOutRate***: What percentage of users opt out from the experimental recommender?
- ***numSession***: How many times users come to use the recommender?
- ***totalLength***: How long do users stay in the recommender (in seconds), which equals to the sum of all their session lengths?
- ***numPageViews***: How many recommendation pages do users browse? Note that we specifically use ***page views*** to refer to recommendation page (with list of movie cards) browsing, not including movie details page view.
- ***numActions***: How many actions do users do in the recommender? ***Action*** refers to either rating, clicking, or wishlisting.
- ***numRatings***: How many items do users rate?
- ***numInterested***: How many of the actions are clicks or wishlistings, which indicate that users are interested in the specific movies that were displayed?

- ***numInterestedPerPage***: How effectively is each recommendation page capturing user interest so that users click the shown items or add them into the wishlist?

Table 5.2: SSA and EXP metrics and their corresponding survey questions.

Metric	**Survey Question**
accuracy	My top-picks match my tastes in movies
familiarity	I am familiar with many of the movies,in my top-picks
diversity	My top-picks have a diverse selection of movies
novelty	My top-picks help me discover new movies
change	I have noticed my top-picks changing
freshness	I like my top-picks for having new recommendations
confusion	I get disoriented sometimes by the change of my top-picks
boredom	I am bored by my top-picks for recommending the same movies
usefulness	My top-picks help me find interesting movies
satisfaction	Overall, I am satisfied by my recent top-picks

Table 5.3: Condition naming for the interaction between cycling and serpentining factors.

Cycling Serpentining	**no**	**intra-session**	**inter-session**
true	*serp.*	*serp-intra.*	*serp-inter.*
false	*ctrl.*	*intra.*	*inter.*

We measure SSA and EXP by embedding a survey into the recommendation page. To have informative responses from users, we set the minimum amount of experience users are required to have in the experimental recommender; specifically, we survey them after they browse ***more than three recommendation pages*** with lists of movie cards. We invite users to respond to the survey by displaying a survey link together with the prominent *recommendation section* title in the page. If users click the link (which is optional), a survey with 10 Likert-scale questions is expanded as listed in Table 5.2. The first eight are measuring SSA, and the remaining two are measuring EXP (*usefulness* and *satisfaction*). Metrics for SSA include four classic metrics used in recommender systems literature: perceived *accuracy*, *familiarity*, *diversity*, and *novelty*. The questions are designed referencing Pu *et al.* 's work on evaluating recommenders through surveys [86, 122]. Because the survey was given while the user interacted with the recommender and to reduce the opt-outs due to a long survey, we chose to implement a short survey that measures each aspect with only one item, rather than

using multiple items per question as proposed by Knijnenburg *et al.* [83]. We also design specific questions pertaining to our manipulation, measuring perceived *change*, *freshness*, *confusion*, and *boredom*. Here the *freshness* question is about the positive aspect of the change, *confusion* asks about the negative aspect of too much change, and boredom is about the negative aspect of too little change. After displaying the first survey, we ask a user the second time with the exact same survey one week later (if they come back to the system and browse for another three or more recommendation pages), in case users do not perceive the recommender dynamics when responding initially.

5.4 Results

We ran the experiment and collected data from March 22, 2016 until May 14, 2016. During this period of time, 6249 users were active in the site. 5158 users were presented with the invitation link to join the experiment and 1218 clicked the link. Overall, 987 users joined the experiment, with each of the 12 (3x2x2) conditions having around 80 users. In subsequent analysis, we also consider the subsample of 802 users who joined the experiment at least half a month before analysis, with each of the conditions having around 66 users. 103 users responded to the surveys, providing 121 responses, 92 of which are complete across all the survey questions.

To verify the randomization, we conducted an analysis on the participants' activity history before they joined the experiment to make sure users across different conditions were comparable. Specifically, we looked at each user's INT metrics during the half month before joining and did not find significant differences.

RQ1: OSA $\Rightarrow$ *INT*. We only consider users who joined the experiment for at least half a month and calculate the metrics for each user's first half month. For users who choose to opt out of the experiment, we exclude activities after the opt-out time. For each metric, we build a negative binomial regression model with the recommender factor as shown in Table 5.3 (six levels) and asking factor (two levels) as the predictors. All models are significantly better than their Poisson regression counterparts (*i.e.*, the data is more overdispersed than what a Poisson model assumes). The results are summarized in Table 5.4. We report results with p-values less than 0.1 here. First, we find that users (including both users who stay and those who opt out) in intra. condition have a higher

probability of opting out than those in ctrl. condition (0.217 vs. 0.094, p=0.005). We do not find significant difference in *numSession* (overall mean is 4.59). On contrary, users in intra. and serp. condition have higher or marginally higher *totalLength* than users in ctrl. condition (3542 vs. 2148, p=0.027; 3147 vs. 2148, p=0.083). The following two metrics *numPageViews* and numActions explain why users in intra. condition spend more time in the recommender. It shows that intra. condition users have marginally higher *numPageViews* (28.1 vs. 21.6, p=0.065), *i.e.*, they browse more; also these users have higher *numActions* (21.7 vs. 11.9, p=0.008), *i.e.*, they do more actions than ctrl. condition users. We find that users in inter. condition have marginally higher *numPageViews* than those in ctrl. condition as well (27.3 vs. 21.6, p=0.098). The following two metrics *numRatings* and *numInterested* further explain which actions users perform more. Consistently with the overall increase of *numActions* for intra. condition users, they not only have higher *numRatings* (15.8 vs. 8.28, p=0.037, *i.e.*, they rate more) but also higher *numInterested* (6.08 vs. 3.70, p=0.031; *i.e.*, they are interested in more recommendations) compared with those in ctrl. condition. Users in serp. and inter. conditions also have higher *numInterested* than ctrl. (6.00 vs. 3.70, p=0.031; 5.77 vs. 3.70, p=0.052), which explains why users in serp. condition spend more time in the recommender. We also separately analyze the metrics for users who stay. They are consistent with the above results and become more statistically significant. Users in either inter., intra., serp. condition have higher *totalLength*, *numPageViews*, *numActions* and *numInterested*. Intra. conditions users also have higher *numInterestedPerPage* which means the probability of those users being interested to click or wishlist is higher compared with users in ctrl. We do not compare across conditions for dropped out users, because it is highly likely that the user population is different.

We also find some interesting effects on *numActions* and *numRatings* for *ask=non-delay* (vs. *delay*) condition that are not included in the table due to space limitations. Specifically, users surveyed in a non-delayed way have higher *numRatings* (12.2 vs. 7.48, p=0.008, *i.e.*, they rate more) and hence have marginally higher *numActions* (16.6 vs. 12.9, p=0.061) than those surveyed in a delayed way.

RQ2: OSA $\Rightarrow$ SSA and EXP. For simplicity of analysis, we only use the 92 *complete* responses for all survey questions to explore this research question. For each metric, we build an ordinal regression (cumulative link mixed effects) model with user

identification as the random intercept (as we have more than one survey response for some users). The fixed effects part of the model has the interaction between *cycling* and *serpentining*, and also *asking* factor as the input. All the results are summarized in Table 5.5. We do not find any significant effects for *asking* and, therefore, it is not included in the table due to space considerations.

First of all, we do not find statistically significant differences between conditions for overall *satisfaction*. However, users in intra., serp. and serp-inter. conditions perceive the recommendations to have less usefulness compared with those in ctrl. condition. This EXP level feedback from users can be explained by comparing the individual SSA metrics. For classic metrics, we find users in inter., intra. and serp. conditions perceive the recommendations to be less accurate than those in ctrl. condition. Similarly, users in intra. and serp. conditions report that they are less familiar with the recommended items.

We also analyze the specially designed metrics for our manipulations. First, we notice that users in inter., intra. and serp-intra. conditions report more perceived change compared with ctrl. condition. This is by design but reassures us that indeed our manipulations are perceived by the users. Given that users perceive the change, a further question regarding the change is whether users like it. In terms of the the positive aspect of the change, users in intra. and serp-inter. perceive significantly more freshness than users in ctrl. condition. Regarding the negative aspects of too much change and too little change, we find users in serp. condition perceive more boredom than those in ctrl. condition. Interestingly, we find users in inter. condition report more confusion than those in ctrl. while users in intra. condition do not perceive significantly more confusion than ctrl. condition.

RQ3: SSA $\Rightarrow$ ***EXP.*** We are interested in how users perceived SSA on recommendations (particularly the ones we specially designed for our manipulations, *i.e.*, change, freshness, boredom, and confusion) affect user EXP (see [83] for the postulated causal relationship between SSA and EXP). To answer this question, we only use all complete survey responses for the 92 users and build two ordinal regression models to predict usefulness and satisfaction with the individual SSA. Table 5.6 shows the results with regression coefficients and p-values. From the table, we find that novelty, accuracy, diversity, and freshness (marginally significant effect for freshness) contribute positively to

user perceived usefulness, in the descending order of effect sizes. Boredom contributes negatively to usefulness although it is marginally significant. While all factors matter for user overall satisfaction except perceived change (which is reasonable, because it measures perception, not preference), the order of the effect sizes is: accuracy, familiarity, boredom, freshness, novelty, diversity and then confusion, where only confusion and boredom contribute negatively and others contribute positively.

5.5 Discussion

Here we discuss the main findings about different top-N list manipulation approaches explored in this work. We find that *intra-session cycling* has an effect of "*scaring*" some users away, while at the same time increasing activity levels for other users, such as browsing more recommendation pages, rating more items, clicking or wishlisting more items, and hence spending more time in the recommender (at least in the first half month we measured). It suggests that this type of manipulation may be a "*love it or hate it*" recommender property. The results obtained via survey questions reveal some potential reasons. In particular, users with this recommender report less perceived *accuracy*, *familiarity*, and marginally less *usefulness*, although they also perceive more *freshness* because of more *change*. Thus, changing recommendations in the same session attracts more user activities but may increase the risk of churning. We hypothesize that the following aspects might be relevant with respect to observed effects and future extensions. First, the platform we use does not have actual item consumption capabilities built in, *i.e.*, users use this movie recommender mainly as a tool to find interesting movies but do not actually watch movies on the site. The dynamics may be quite different in platforms with consumption, because users can proceed with item consumption directly after a recommendation instead of speculating or processing the recommendation as a piece of information to be used later. The increasing effect of user activities should be interesting to system designers, but further study is needed to explore to what extent this effect generalizes to platforms with built-in consumption. Second, cycling 240 items (*i.e.*, the value of N in our top-N recommendations) in our study may represent too big of a range for some users. They may experience dramatic accuracy degradation after cycling for a while, which could contribute to their opt-out. Thus, testing the cycling

approach with smaller values of N in different platforms also constitutes an interesting topic for future research.

Inter-session cycling and serpentining are the two best-performing conditions in our experiment, considering both opt-out rate and user activities. They do not have a significant effect of "*scaring*" users away, especially the serpentining approach. At the same time, both of them increase user activities such as clicking or wishlisting, especially for users who stay (*i.e.*, do not opt out). The results also show a trend that, in these conditions, users who stay are more active, while users who stay in the control condition are less active compared with those who opt out. This suggests that we are able to retain more active users through our top-N list manipulations. However, we also want to point out that users with *inter-session-cycled* recommendations report less *accuracy*, more *change*, and also more *confusion*. Users in a serpentining recommender also report less *accuracy*, *familiarity*, and more *boredom*. Interestingly, users in *inter-session cycling* instead of *intra-session cycling* perceive more *confusion* than those in the control condition. This might result from the fact that users perceive the change of recommendations but cannot connect the change with their own previous activities when they come back to the system in the next visit. This suggests that users demand at least certain extent of control (or sense of control) in using a recommender system. They expect the recommendations to change based on their taste or at least what they tell about their taste to the recommender.

We see interesting interactions between *cycling* and *serpentining*. *Serpentining* mitigates the negative effects of cycling, such as opt-out rate for intra-session, reduction in *accuracy*, and increased *confusion* for inter-session. However, *serpentining* also reduces the positive effects of cycling, such as increased user activities and improved *freshness* for intra-session. One exception is that the effect of inter-session cycling with serpentining on perceived freshness is positive. According to these results, it seems that combining the two manipulations makes things too complicated for users to build a mental model on how the recommender is working.

Although the interface of MovieLens has a grid-view layout, we believe that these approaches are generalizable to other layouts, such as lists. Even if a list does not have pagination, our algorithms can be adapted by using the top-N as the length of the list, which can be *serpentined* and *cycled*, although such manipulations might be more

apparent from the user's perspective (*e.g.*, if the list is short).

We would like to note to two limitations in our study: self selection bias and uncertainty about longer term effects. First, our analysis shows that users who were qualified for the experiment but chose not join were significantly less active than users who joined the experiment. Second, the duration of this experiment does not permit us to draw conclusions about longer-term usage patterns, either for those to retain *serpentined* and/or *cycled* recommendations or from those who experience them but opt out.

Table 5.4: Results of different conditions for INT metrics. *au* indicates the measurement and effect across all users in that treatment group, including users who opt-out, returning to their default recommender. *su* indicates the measurement and effect for users who retain the experimental recommender in the measured first half month. We include both to estimate the effects both on those who retain the treatment and on the population of users offered the treatment overall. We analyzed users who opt-out separately, and in no measurement did they differ significantly from the control group. *numSessions* (overall mean is 4.59) is not shown because there were no statistically significant differences. See Table 5.3 for the definition of condition names that combine *cycling* and *serpentining*. The numbers are means with standard errors in the parentheses and only significant comparisons (through *negative binomial regressions*) are marked with significance codes: + ($p < 0.1$), * ($p < 0.05$), ** ($p < 0.01$).

		Conditions combining cycling and serpentining					
		ctrl. (n=148)	**inter. (n=134)**	**intra. (n=129)**	**serp. (n=145)**	**serp-inter. (n=128)**	**serp-intra. (n=118)**
optOutRate		0.094 (0.024)	0.149 (0.030)	0.217 (0.036) >ctrl. **	0.117 (0.026)	0.132 (0.029)	0.093 (0.026)
totalLength	***au***	2148 (333)	3107 (506)	3542 (588) >ctrl. *	3147 (493) >ctrl. +	2688 (448)	2153 (374)
	su	2148 (344)	3369 (585) >ctrl. +	3442 (635) >ctrl. +	3314 (543) >ctrl. +	2647 (466)	2127 (387)
numPageViews	***au***	21.6 (2.09)	27.3 (2.76) >ctrl. +	28.1 (2.90) >ctrl. +	24.9 (2.43)	19.6 (2.04)	21.3 (2.31)
	su	20.6 (2.06)	29.5 (3.18) >ctrl. *	27.8 (3.19) >ctrl. *	26.5 (2.70) >ctrl. +	18.9 (2.08)	20.5 (2.30)
numActions	***au***	11.9 (1.87)	15.2 (2.49)	21.7 (3.62) >ctrl. **	14.5 (2.28)	10.3 (1.74)	15.4 (2.70)
	su	10.5 (1.70)	16.6 (2.91) >ctrl. +	19.5 (3.63) >ctrl. *	15.7 (2.60) >ctrl. +	8.45 (1.51)	15.9 (2.87) >ctrl. +
numRatings	***au***	8.28 (1.77)	9.61 (2.16)	15.8 (3.63) >ctrl. *	8.68 (1.88)	6.04 (1.39)	11.1 (2.68)
	su	7.35 (1.64)	10.5 (2.55)	13.6 (3.50) >ctrl. +	9.70 (2.21)	4.43 (1.09)	11.6 (2.91)
numInterested	***au***	3.70 (0.588)	5.77 (0.952) >ctrl. +	6.08 (1.02) >ctrl. *	6.00 (0.951) >ctrl. *	4.35 (0.740)	4.35 (0.771)
	su	3.21 (0.538)	6.26 (1.11) >ctrl. **	6.15 (1.16) >ctrl. *	6.25 (1.04) >ctrl. **	4.07 (0.742)	4.32 (0.801)
numInterestedPerPage	***au***	0.137 (0.022)	0.148 (0.025)	0.204 (0.036)	0.160 (0.026)	0.168 (0.029)	0.193 (0.035)
	su	0.132 (0.022)	0.153 (0.028)	0.214 (0.042) >ctrl. +	0.148 (0.026)	0.156 (0.029)	0.188 (0.036)

Table 5.5: Results of different conditions for SSA and EXP metrics. See Table 5.3 for the definition of condition names that combine *cycling* and *serpentining*. ctrl. condition is the base to compared with in the ordinal regressions. The numbers are coefficients (in log odd-ratio scale) with standard errors in the parentheses. Significance codes: + ($p < 0.1$), * ($p < 0.05$), ** ($p < 0.01$).

	inter.	**intra.**	**serp.**	**serp-inter.**	**serp-intra.**
satisfaction	-0.698 (0.707)	-0.849 (0.662)	-0.863 (0.768)	-0.482 (0.740)	-0.440 (0.757)
usefulness	-0.793 (0.774)	-1.19 (0.719) +	-1.99 (0.866) *	-1.72 (0.799) *	-1.26 (0.804)
accuracy	-1.72 (0.751) *	-1.94 (0.715) **	-1.51 (0.828) +	-0.477 (0.822)	-1.08 (0.785)
familiarity	-0.731 (0.782)	-1.47 (0.748) *	-2.19 (0.911) *	-0.453 (0.811)	-0.940 (0.850)
diversity	-0.037 (0.737)	0.999 (0.705)	-0.823 (0.843)	0.874 (0.793)	0.413 (0.786)
novelty	-0.449 (0.698)	-0.201 (0.659)	-1.22 (0.779)	-0.777 (0.761)	-0.673 (0.772)
change	2.78 (1.00) **	2.37 (0.868) **	0.668 (0.912)	1.16 (0.869)	2.02 (0.954) *
freshness	0.903 (0.741)	1.63 (0.720) *	0.437 (0.826)	1.60 (0.816) +	1.08 (0.806)
boredom	-0.454 (0.803)	0.319 (0.763)	1.89 (0.966) +	1.08 (0.891)	1.11 (0.871)
confusion	1.87 (0.895) *	1.10 (0.792)	-0.827 (0.963)	0.163 (0.865)	1.28 (0.888)

Table 5.6: The coefficients and standard errors (in parentheses) of the ordinal regressions using individual SSA to predict EXP (usefulness and satisfaction). Significance codes: + ($p < 0.1$), * ($p < 0.05$), ** ($p < 0.01$), *** ($p < 0.001$).

SSA	EXP	
	usefulness	*satisfaction*
accuracy	1.10 (0.29) ***	1.54 (0.31) ***
familiarity	-0.160 (0.24)	0.807 (0.26) **
novelty	1.22 (0.29) ***	0.661 (0.28) *
diversity	0.613 (0.27) *	0.569 (0.26) *
change	-0.204 (0.31)	0.018 (0.28)
confusion	-0.093 (0.28)	-0.479 (0.28) +
freshness	0.498 (0.30) +	0.702 (0.29) *
boredom	-0.466 (0.26) +	-0.778 (0.27)**

Chapter 6

Optimizing for User Interaction

6.1 Introduction

Machine-learning-based recommender systems are driven by user feedback data, *e.g.*, explicit feedback data of ratings [28] and implicit feedback data of actions [72]. Typically, supervised learning models to predict rating values or action probabilities are trained in these systems based on the theory of empirical risk minimization [123], *i.e.*, optimizing to reduce the prediction errors in historical training data with regularization to maintain generalizability.

In the history of recommender system research, there was a transition of trend from using explicit feedback data to implicit feedback data. The earlier pursuit of the Netflix Prize [28] was a (explicit-feedback) rating prediction problem. It greatly drove the progress of recommender system research. Amatriain and Basilico [124] in Netflix Blog later pushed back on researcher's focus on rating prediction, arguing that "accurate prediction of a movie's rating is just one of the many components of an effective recommendation system" and "using predicted ratings on their own as a ranking function can exclude items that the member would want to watch even though they may not rate them highly". They turned to a broad set of techniques to model the various types of user action data in the system to recommend items that the member is most likely to play and enjoy.

Until today, the difference between recommender systems built on explicit vs. implicit feedback data is not addressed in the research literature. This type of research

inquiry is hard because offline metrics can not address it while live experiments require access to platforms that have real active users. We set out to compare the two types of recommenders in a live experiment in MovieLens based on the classical matrix factorization algorithms [18]. Being aware that there are many different algorithms proposed for modeling both explicit and implicit feedback data [18, 72, 111, 110], we chose the classical ones for better controlling the differences of algorithms while focusing on studying the differences that are inherent in the two types of feedback data. This comparison is possible also because the majority of the activities that users perform on the site is rating movies, which gives us access to not only implicit user actions but also abundant explicit ratings.

User-centric research in recommender systems [82] tends to focus on a broad set of factors and metrics that contribute to the success of recommender systems with the theoretic goal of supporting user decision making processes in front of a large amount of choices. Particularly, accuracy, diversity, novelty, serendipity, popularity, freshness and recency etc. [86, 82, 105, 112, 125] have all been studied in prior literature. Within this framework, accuracy of predicting ratings or actions only reflects part of the many factors that are important. Previous research has shown that diversification, blending in popularity etc. on top of predictions help improve user engagement and experience [112, 125]. However, how exactly to blend multiple factors in to produce a final set of recommendations is tricky because there is no single ground-truth objective to target (*e.g.*, the success of recommender systems is too abstract) to guide the blending process.

Adomavicius *et al.* [12] define the recommendation problem as a multi-criteria decision making problem (MCDM) and argue that the suitability of a recommendation for a particular user may depend on more than one utility-related aspect that the user takes into consideration when making the choice. Correspondingly in machine-learning-based recommender systems, it is typically modeled as a multi-objective optimization problem to take into account the multiple criteria [13]. However, the question that whether this approach can always help achieve an optimal solution across all criteria or whether it compromises some criteria while improving others is not yet fully understood. Online tuning in live systems is necessary to find the best combination weights but it usually takes long cycles to tune and hence is very expensive to follow.

In this research, we tried two new approaches in a live experiment to combine multiple factors in a principled way (also using machine learning techniques) inspired by the social theory of technology acceptance [87] and the reinforcement learning theory on decision making under uncertainty [126]. The first approach is to target user return (*i.e.*, technology acceptance) as the objective to combine multiple factors. The second approach is to target online (on-policy in decision-making terms, see the related work section) user interaction following an online learning (specifically contextual bandit learning) algorithm.

The above motivation leads us to the following two research questions.

- *RQ1: What are the differences between recommender systems based on explicit vs. implicit user feedback data modeled with the classical matrix factorization algorithms?*
- *RQ2: Do multi-factor-blending recommendation algorithms lead to improved or changed user experience and if so, how?*

In the rest of the chapter, we report on a live experiment involving more than 1.5K real users of a movie information site using six different recommenders for at least one month, measuring a broad set of user-centric metrics including objective user activities and subjective user perceptions. We found that

- a recommender based on a matrix factorization model minimizing (implicit) action prediction error engages users more (in terms of page views and interactions with the recommendations) than a matrix factorization model minimizing (explicit) rating prediction error, which empirically explains the transition from modeling explicit feedback data to implicit feedback data in recommender system research.
- the increased positive engagement is also associated with a significant amount of increase in user negative engagement (*e.g.*, low ratings, clicking "not interested", browsing effort), likely because implicit feedback is noisier than explicit feedback about user preferences.
- blending both explicit and implicit feedback from users by targeting online (on-policy) user interaction through a contextual bandit learning algorithm can help

gain the benefits of engagement and mitigate the possible cost, although it does not further significantly drive engagement.

- with our current design, targeting user return as the objective does not significantly affect user engagement (*e.g.*, the actual future user return and churning risk) and shows a trend of hurting perception metrics compared with the baseline.

In the following sections, we first introduce the necessary background on user-centric research and machine learning techniques in recommender systems. The used techniques of this work span from classical matrix factorization (together with stochastic gradient descent) to contextual bandit learning (particularly the LinearUCB algorithm) and the Q-learning algorithm. Then we detail the method of this work and elaborate the online field experiment design. We show the results next and discuss the findings along with potentially promising future work. Lastly, we summarize this work's conclusion and contribution.

6.2 Background of User-Centric Research

User-centric research in recommender systems has been increasingly important. As pointed out by McNee *et al.* [81], recommendation accuracy on its own often is not a sufficient indicator of recommendation quality, and further work by Konstan *et al.* [82] elaborates the evolution of recommender system research from being concentrated purely on algorithms to research focused on the rich set of questions around the user experience with the recommender.

Several frameworks have been proposed and widely used by researchers to evaluate and understand user experience in recommender systems. For example, Knijnenburg *et al.* [83] proposed a comprehensive framework taking into account both objective system measurements and subjective user perceptions to explain user experience. McNee *et al.* [85] proposed an analytical process model called Human Recommender Interaction that acts as a bridge between user information seeking tasks and recommendation algorithms to help with the design and structure of recommender systems. Pu *et al.* [86] proposed a user centric evaluation framework by employing state-of-the-art survey designs structured and derived based on theories of human behavioral intention and reasoned action. Particularly relevant to our research, the theory of UT-AUT developed

by Venkatesh *et al.* [87] postulates important social user factors that cause people to develop behavioral intention towards technologies and actual behavior of accepting or abandoning of the technologies. We are interested in studying whether this theory can be combined with machine learning, especially with reinforcement learning techniques, to optimize for user acceptance of recommender systems at scale. Work from Xiao *et al.* [88] is a direct application and further development of this theory in the domain of e-commerce recommender agents, *e.g.*, highlighting the importance of trust, perceived ease of use, and perceived usefulness in determining the user intention of future use of the recommender agents.

6.3 Background of Machine Learning

To support explaining the recommenders we built in the experiment, this section sets up notations and gives a formal background on machine learning techniques used in this work. Note that this section is not meant to give an overview on how machine learning can be applied in recommender systems (See [127, 128] for a better review). Instead, it serves the purpose of motivating our research and explaining the perspective of approximating recommendation as a statistical learning problem.

Denote u as the representation of a user (*e.g.*, basic profile, history interactions with the system) and c as the current context (*e.g.*, time, location etc.) of a user entering a recommender system requesting item recommendations. Define $s = (a; c)$, *i.e.*, s describes the the environment or state of both the user and the system. Denote a to be the action or decision that a recommender system needs to make. In the most simple case, a might be an item to recommend or a set of recommendations to present. More broadly however, it might incorporate the decision of how to present.

6.3.1 Empirical Risk Minimization

In the theory of supervised-learning-based recommender systems, it assumes that there is a y that represents u's preference under context c on a and it follows an unknown conditional distribution $P(y|s; a)$ (*i.e.*, we focus on determinant statistical models here instead of generative models; see [129] for the difference). If we can reliably estimate this distribution for all possible s and a (*e.g.*, sufficient observations are made) and fix our

decision policy of making recommendations to always pick a with the largest $E(y)$, then the recommendation problem becomes the following stochastic optimization problem, where L an objective function measuring the loss or error of estimating y with a model (or function) f and $E_P(x)$ denotes the expectation of x with respect to the distribution P (also called the Statistical Decision Theory for supervised learning [130]).

$$f^* = argmin_f E_P(L(f, y)) \tag{6.1}$$

In reality, since we do not know the true P (assuming there exists such a P), the theory further assumes that the observed user feedback data are I.I.D samples of P. Since parametric models are popularly used as f in recommender systems, *e.g.*, the widely used and studied matrix factorization [18], we denote f with $f(W, x)$ where $x = (s; a)$ and W are all model parameters without loss of generality. Therefore, the problem in Equation 6.1 is further simplified to the following Empirical Risk Minimization problem [123] (Equation 6.2), where N is the number of observations (x, y pairs) from user feedback data. $g(w)$ is a regularization term used to penalize large W, *e.g.*, in terms of L_1 or L_2 norms and is the key of W^* having theoretical guarantees when generalizing to the unknown P [123]. λ is a scalar parameter controlling the strength of the penalty.

$$W^* = argmin_w \sum_{i=1}^{N} L(f(W, x_i), y_i) + \lambda g(W) \tag{6.2}$$

6.3.2 Matrix Factorization

Matrix factorization is a type of low-dimensional embedding model where $x = (u; c; a)$ are represented by low-dimensional dense vectors. The dimension noted as d here is a hyper-parameter. FunkSVD [7, 18] is a basic version of the family of matrix factorization models used in this work. See SVDFeature [21] and libFM [110] for generalized versions. Specifically following Equation 6.2, we have

$$f(W, x_i) = f((b, U, V), (u_i; a_i)) = b_0 + b_{u_i} + b_{a_i} + U_{u_i}^T \cdot V_{a_i} \tag{6.3}$$

Among the model parameters $W = (b, U, V)$, b is a bias vector and U and V are factor matrices. Note that we use u and a as the index into b, U and V. b_0 is reserved for the global bias (similar to the intercept in linear regression models).

Depending on the domain of y, different loss functions L are suitable. For a rating prediction problem, we use a least-squares loss function as follows.

$$L(f, y) = \frac{1}{2}(f - y)^2 \tag{6.4}$$

For an action prediction problem, assuming $y \in \{0, 1\}$ representing whether there is a positive action or feedback from the user or not. Define the sigmoid function $\sigma(f) = 1/(1 + exp(-f))$

$$L(f, y) = y \ln \sigma(f) + (1 - y) \ln(1 - \sigma(f)) \tag{6.5}$$

For both types of problems, we use L_2 norm regularization. That is,

$$g(w) = \lambda_1 |b|_2^2 + \lambda_2 |U|_2^2 + \lambda_3 |v|_2^2) \tag{6.6}$$

The Stochastic Gradient Descent (SGD) [18] algorithm (shown in Equation 6.7) is widely used to solve the optimization problem in Equation 6.2 when f is a matrix factorization model. Define $err = y - \hat{y}$ where $\hat{y} = f$ for rating prediction and $\hat{y} = \sigma(f)$ for binary action prediction. Denote η as the learning rate. The SGD algorithm follows the following updating rules. Note that the updating rule for b is common for b_0, b_u and b_a.

$$\begin{aligned} b &\leftarrow b + \eta(err - \lambda_1 b) \\ U_u &\leftarrow U_u + \eta(err \cdot V_a - \lambda_2 U_u) \\ V_a &\leftarrow V_a + \eta(err \cdot U_u - \lambda_3 V_a) \end{aligned} \tag{6.7}$$

6.3.3 Q Learning

Different from supervised learning, when applying the theory of reinforce learning [131, 126] in recommender systems, the recommendation problem is modeled as a sequential decision-making problem. At any time step t ($t = 1, ..., T$) where T is the horizon to consider, the system is faced with the decision of taking action a_t given a state s_t, *i.e.*, $(u_t; c_t)$. For each a_t that the system takes, it is given a reward feedback r_t which could be proportionate to the user's rating or whether the user performs positive action on the

item recommendation. The goal of the system is to find a policy, which is a mapping function $\pi(s) \rightarrow a$, to maximize its accumulative reward across the horizon T, *i.e.*, $argmax_{\pi} R_T$ where $R_T = \sum_{t=1}^{T} r_t$. This accumulative reward is called the value of a policy π (*Policy Value*) if it is followed across the horizon. The *Reward Maximization* problem is unbounded when T goes to infinity unless a discounting factor is applied for future rewards, *e.g.*, defining $R_T = \sum_{t=1}^{T} \gamma^{t-1} r_t$ where the *Discounting Rate* $\gamma \in [0, 1]$.

The full reinforcement learning problem requires learning not only the reward function $r(s, a)$ but also how the environment or state might change because of its action, *i.e.*, an unknown transition distribution $P(s_{t+1}|s_t, a_t)$. Define $Q(s, a)$ as the accumulative reward or value of taking action a in state s and then following the best policy π' afterwards. The algorithm that iteratively estimates $Q(s, a)$ (instead of $r(s, a)$) according to the following Bellman equation is called *Q-learning*, where s' is the next possible s after taking action a in state s. Q-learning solves the reinforcement learning problem with the solution $\pi'(s, a) = argmax_a Q(s, a)$

$$Q(s, a) = r(s, a) + \gamma \sum_{s'} P(s'|s, a) argmax_{a'} Q(s', a') \tag{6.8}$$

In applying Q-learning in real-world problems, we usually assume a parametric form for the Q function $Q(s, a)$. *e.g.*, in [40], an ensemble of trees are used. In this work, a simple linear function is used (see the Method section). At the beginning of running the algorithm, the parameters of Q is randomly initialized based on which the right-hand side of Equation 6.8 can be calculated. Then $Q(s, a)$ adapts itself (by changing parameters) to fit the right-hand side value. This process is iteratively done until $Q(s, a)$ converges.

6.3.4 Regret Minimization

If we make the assumption that a does not have an effect on the state s, *i.e.*, s_t is I.I.D samples of an unknown distribution $P(s)$, the reward maximization problem can be converted to the problem of minimizing the regret of a policy compared with the best policy π' in an assumed family of policies, *i.e.*, the following *Regret Minimization* [132]

problem:

$$\pi^* = argmin_\pi \sum_{t=1}^{T} (r(s_t, a_{\pi',t}) - r(s_t, a_{\pi,t})) \tag{6.9}$$

Following is a list of the key theoretical differences between the Regret Minimization problem (shortened as RM) in Equation 6.9 and the Empirical Risk Minimization (shortened as ERM) problem in Equation 6.2.

- The RM problem assumes $s_t \sim_{i.i.d} P(s)$ while the ERM problem assumes $x_t \sim_{i.i.d} P(x)$ where $x = (s; a)$.
- The algorithm to solve the RM problem is through online learning. That is, learning as t goes from $t = 1$ to T (effectively reading data once) while the algorithm to solve the ERM problem is iterative typically by reading the observation data multiple times, *e.g.*, the SGD algorithm.
- There is a difference in terms of *On-policy* vs. *Off-policy* for the algorithm of the RM problem while the algorithm of the ERM problem does not have such as difference. This distinction actually derives from the first one. When assuming $x_t \sim_{i.i.d} P(x)$, the distribution of the observations is fixed although unknown. In contrary, if only $s_t \sim_{i.i.d} P(s)$ is assumed, different policies or algorithms can observe different distributions of a_t, $r(s, a)$ (which is closer to reality) and hence learn differently. This makes the evaluation of the algorithm for the RM problem hard because the ideal evaluation (many off-policy evaluation methods have been proposed in the literature with certain limitations [133]) takes actual running of the algorithm or policy in the system, *e.g.*, through live experiments, which is done in this research.

6.3.5 Contextual Bandit and LinearUCB

The Contextual Bandit [35] has the same objective as the regret minimization problem in Equation 6.9. In our notation, $s = (u; c)$ is considered as the context and both s and a are represented as feature vectors (contrary to the discrete value space in the Multi-Armed Bandit [35] problem), *e.g.*, the vectors of user profile or item attributes. A well-studied algorithm for the contextual bandit problem which is also used in this research

is LinearUCB [35]. LinearUCB makes further assumptions about $r(s, a)$ to simplify the problem in Equation 6.9. Specifically, it assumes $(r|x = (s; a)) \sim Gaussian(x^T\theta, \sigma)$, *i.e.*, the expected reward is linearly related to the input feature vectors. For the purpose of explaining the method of this research, the LinearUCB algorithm is elaborated as follows in Algorithm 1 with the notations in this chapter. See [35] for a theoretical bound of the regret of this algorithm.

Algorithm 1: The LinearUCB algorithm

Data: A sequence of $s = (u; c)$ while the algorithm is running in the system. The system has M possible actions (items), each of which represented by a feature vector a. Denote $x_{t,k}$ to be $(s_t; a_k)$ for $k = 1, 2, ..., M$.

Parameters:α, β, d is the dimension of $x_{t,k}$

Initialization: $A = \beta I_d$, $h = 1_d$, *i.e.*, equal weights for all input features.

1 **for** $t = 1, 2, ..., T$ **do**
2 $\quad \theta_t = A^{-1}h$
3 $\quad$ **for** *k=1, 2, ..., K* **do**
4 $\quad\quad \hat{r}_{s_t,a_k} = x_{t,k}^T\theta_t + \alpha\sqrt{x_{t,k}^T A^{-1} x_{t,k}}$
5 $\quad$ **end**
6 $\quad$ Take action $a_t = argmax_k \hat{r}_{s_t,a_k}$, let $x_t = (s_t; a_t)$
7 $\quad$ Receive actual reward feedback r_t
8 $\quad A \leftarrow A + x_t x_t^T$
9 $\quad h \leftarrow h + x_t r_t$
10 **end**

Result: θ_T

Note that in real systems, a page of items will be recommended at one time t instead of a single-item action. In this work, we update Step 6 in Algorithm 1 to take K actions or items $a_1, a_2, ..., a_K$ at the same time t and we observe the user feedback for all items and then Step 8-9 are executed for each of them.

6.4 Method

We conducted an online field experiment on MovieLens to answer our research questions. To better understand the effects of recommendation algorithms on user engagement, we

define the following two categories of user actions on recommendations.

- **Positive Actions** are defined as high ratings (rating¿=4.0), clicking to see details of movies or adding movies into a wishlist.
- **Negative Actions** are defined as low ratings (rating¡=3.5) or clicking the "not interested" icon.

The experiment follows a between-subjects design, *i.e.*, a user is randomly and persistently assigned into one of the six recommenders in Table 6.1 (all users need to sign in to use the site features). During the experiment, when MovieLens users visit the front page, we display a prominent invitation link at the top asking users whether they would like to experience a new recommender. If they click the link, an informed consent page is displayed where we briefly introduce the purpose of the study (not the experiment details). Users can either accept or decline to participate in the experiment. If the user accepts, we randomly assign the user persistently through the experiment into one of the six recommenders. After that, this user's site browsing is powered by the assigned recommender, including the item display in the top recommendation section of the front page, the recommendation explore page (potentially with additional user specified filters, *e.g.*, genres or release dates). Users can click a link at the top right corner of the site to opt out from the experimental recommender anytime going back to their original recommender. If a user chooses to opt out, the user cannot go back into the experiment anymore. This study was approved by the Institutional Review Board of our organization.

6.4.1 The Six Recommenders

Table 6.1 lists the six recommenders we build for the experiment. For all of the LinearUCB algorithms, we set the exploration parameter $\alpha = 0$ and prior $\beta = 1$ (since there is natural exploration because we make a page of recommendations at once, we leave the study of the exploration effect as future work). For the latter four recommenders (Bandit-* and Reinforce-State), besides predictions made by MF-Rating and MF-Action, another two factors from item attributes are introduced: *Recency* and *Popularity* as defined in the following list. In order to fairly combine these four factors at

Table 6.1: The six recommenders studied in this work

Recommender	Input	Output	Model	Algorithm
MF-Rating	u: user ID a: item ID	y: rating	Equation 6.3 for f Equation 6.4 for L	SGD in Equation 6.7
MF-Action	u: user ID a: item ID	y: positive action or not	Equation 6.3 for f Equation 6.5 for L	Same as MF-Rating
Bandit-Two	$a = (a_1; a_2)$ a_1: predicted rating a_2: predicted action	r: positive action or not on-policy	$\theta = (\theta_1; \theta_2)$ $E(r(a)) = a^T\theta$	LinearUCB in Algorithm 1
Bandit-Four	$a = (a_1; ...; a_4)$ a_1, a_2: Same as Bandit-Two a_3: recency of item ID a_4: popularity of item ID	Same as Bandit-Two	$\theta = (\theta_1; ...; \theta_4)$ $E(r(a)) = a^T\theta$	Same as Bandit-Two
Bandit-State	$x = (s; a)$ a: Same as Bandit-Four $s = (s1; ...; s4)$ $s_i = \frac{1}{N_u}\sum_{j=1}^{N_u} a_{i,j}$, for $i = 1, ..., 4$	Same as Bandit-Four	$\theta = (\theta^{(a)}; \theta^{(s)})$ $E(r(s,a)) = \sum_{i=1}^{4} a_i \cdot \theta_i^{(a)}$ $+a_i \cdot s_i \cdot \theta_i^{(s)}$	Same as Bandit-Four
Reinforce-State	Same as Bandit-State	$r_t = 1$ if the user returns in a week and t is the end of a session $r_t = 0$ otherwise	$\theta = (\theta^{(a)}; \theta^{(s)})$ $Q(s,a)$ is in Equation 6.10	Q-learning in Algorithm 2

the beginning (since they are in different scales), we map them into percentiles looking at one year of historical recommendation data. By default, these four factors refer to percentiles instead of the raw values in the rest of the chapter.

- *predicted rating*: the predicted rating of MF-Rating for a user ID on an item ID.
- *predicted action*: the predicted positive action probability of MF-Action for a user ID on an item ID.
- *recency*: the release year of the movie item
- *popularity*: the total number of ratings on the movie in the system

Reinforce-State recommender is inspired by the theory of UT-AUT [87] in which the ultimate goal of a technology could be optimizing for user adoption or acceptance. While whether we can directly optimize for user acceptance is an unanswered question, we set out to approach it by employing Q-learning techniques that can handle delayed reward feedback. Reward here becomes whether the user returns within a certain period of time (one week is used in this work). That is, whether a user's session with a recommender system is good or bad is determined by whether the user will come back with a new session within the coming week, as illustrated in Figure 6.1. The same linear function is used as in Bandit-State recommender to approximate Q function for the purpose of

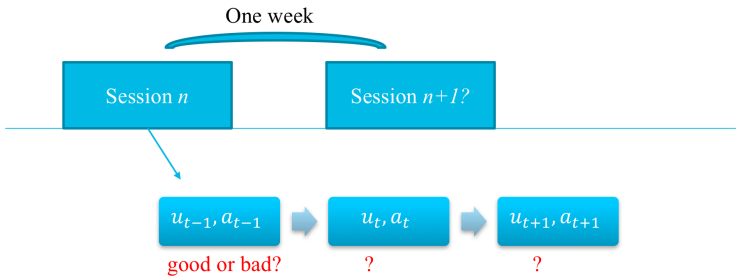

Figure 6.1: An example illustration of utilizing delayed user feedback *user return* as the learning signal.

fair comparison across recommenders as in the following Equation 6.10. Note that it also models the interaction effects of a and s.

$$Q(s,a) = \sum_i a_i \cdot \theta_i^{(a)} + a_i \cdot s_i \cdot \theta_i^{(s)} = \sum_i a_i \cdot (\theta_i^{(a)} + s_i \cdot \theta_i^{(s)}) \tag{6.10}$$

We also need to model user state and its transition in order for the Bellman Equation 6.8 to run (*i.e.*, we employ a model-based Q-learning instead of model-free [126]). From the theory UT-AUT [87], important psychological metrics characterize the user state, which however is unobservable. The PO-MDP [36] model is designed for this case, but to have better experimental control, tractability and model interpretation, we use a deterministic state transition model. As shown in Table 6.1, we characterize the current user state s as the aggregate means of the four involved factors calculated on all historically recommended items (with size N_u) to the user u. Then the state transition is straightforward when we make a new recommendation as follows in Equation 6.11 (*i.e.*, re-calibrating the means). Based on this transition model and the Bellman Equation 6.8, Algorithm 2 shows in details how we train the Reinforce-State model targeting the delayed reward user return. In the algorithm, Steps 8-9 are equivalent to a LinearUCB update step with exploration parameter $\alpha = 0$, *i.e.*, using the LinearUCB algorithm as an online algorithm to estimate the linear Q function. Figure 6.2 shows the learning process of the Reinforce-State model. The shape of the curve shows a trend of convergence after 20 epochs. It also reflects that this recommender tends to recommend items that after being displayed users have more sessions afterwards. Note that Q-learning is an off-policy algorithm and hence can be trained on historical data although the quality

of the estimated Q values depend on how well the state and action spaces are already explored in the data set.

$$s'_i(s_i, a_i) = \frac{s_i \cdot N_u + a_i}{N_u + 1}, i = 1, ..., 4 \tag{6.11}$$

Algorithm 2: The Q-learning algorithm for Reinforce-State recommender.

Data: A sequence of $(userID_t, itemID_t, return_t)$ where $t = 1, ..., T$. $return_t$ is pre-calculated by organizing the historical data as sessions and setting $return_t = 1$ if t is the last recommendation of a session and the user has another session within the coming week. Otherwise, $return_t = 0$

Parameters: The maximum number of rounds: $epoch_{max}$; learning rate η; the number of features $d = 8$ here

Initialization: Initialize $A = \alpha I_d$, $h = \beta 1_d$; Pre-train the models of MF-Rating and MF-Action

for $epoch = 1, 2..., epoch_{max}$ **do**
  for $t = 1, 2, ..., T$ **do**
    Calculate $(s_t; a_t)$ by making predictions on $(userID_t, itemID_t)$ using MF-Rating and MF-Action models.
    Calculate s' based on Equation 6.11.
    $\theta = (\theta^{(s)}; \theta^{(a)}) = A^{-1}h$,
    Calculate $Q(s', a)$ for all possible a based on Equation 6.10 (which involves making predictions for $userID_t$ on all item IDs with MF-Rating and MF-Action models to get a)
    Calculate the right-hand side of the Bellman Equation 6.8. Denote the value as Q'_t.
    $A \leftarrow A + (s_t; a_t)(s_t; a_t)^T$
    $h \leftarrow h + (s_t; a_t)Q'_t$
  end
end

Result: $\theta_{epoch_{max},T}$

The models of MF-Rating, MF-Action and Reinforce-State are trained offline, while the three models of contextual bandits learn online (on-policy). During the experiment,

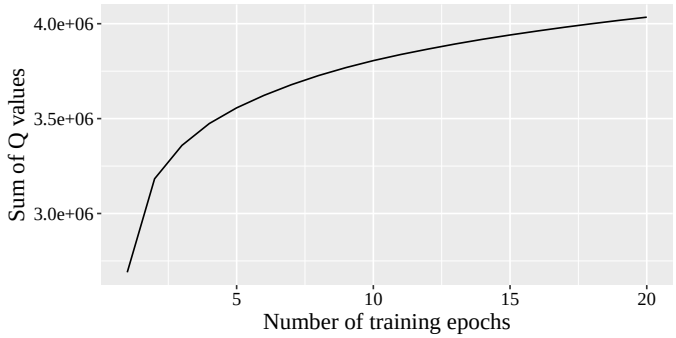

Figure 6.2: The Q-learning curve in training the reinforcement delayed reward model using one year of historical data. The y-axis is the sum of Q-values for all $t = 1, ..., T$ in each epoch on the x-axis.

we train the three offline models in batch every week using the past one year of historical data (around 52M movie displays with 2M positive actions and 2.1M ratings; note that the majority of user activities in the site are ratings). We see accuracy gains when using recent one year of data compared with using all historical data evaluating in a temporal way (*i.e.*, training on historical data excluding the most recent week and testing on this most recent week of data). The accuracies on average are 0.957 in testing RMSE (Root Mean Squared Error), 0.735 in testing MAE (Mean Average Error) for the rating prediction model, and 0.703 in testing AUC (Area Under the ROC) for the action prediction model.

When training the two models each week during the experiment, we use the recent week of data as the validation set to avoid over-fitting (regularization parameters for both bias and factor terms in Equation 6.3 are set to 1e-5, 20 epochs at maximum, 30 latent dimensions after tuning offline). Then we further update the trained models by running the SGD steps in Equation 6.7 over the validation data once to make sure the models are up to date. During the week before the next batch training, the models are updated in real-time by running the same SGD steps once whenever the users rate or browse movies in the site.

6.4.2 Objective Measurements

We measure the following objective activity metrics. In order to have better controlled observations, we set an observation time window for one month, *i.e.*, we only look at each user's activities for one month after the user joins the experiment. We exclude users who joined the experiment too recently to collect data across the full time window. If a user opts out from the experiment during the observation time window, we cut off the data collection after the moment that the user opts out.

- *#sessions*: the number of sessions, *i.e.*, how many session-level visits to the site do users have. Each user has one observation for this metric.
- *#frontView*: the number of front page views. Each user has one observation for this metric.
- *#exploreView*: the number of explore page views. Each user has one observation for this metric.
- *frontPositive*: whether there is any positive action on the eight(top-K=8) recommendations in the front page. Each front page view has one observation for this metric.
- *frontNegative*: whether there is any negative action on the eight(top-K=8) recommendations in the front page. Each front page view has one observation for this metric.
- *explorePositive*: whether there is any positive action on the recommendations (top-K¿=24) in the explore pages. Each explore page view has one observation for this metric.
- *exploreNegative*: whether there is any negative action on the recommendations (top-K¿=24) in the explore pages. Each explore page view has one observation for this metric.
- *optOut*: whether a user opts out from the assigned recommender during the time window. Each user has one observation for this metric.

6.4.3 Subjective Measurements

We also deploy surveys with the following list of question items asking users about several perceptional aspects of the recommenders. For some of the classical metrics, *e.g.*, accuracy, diversity, novelty etc., we directly use the design of the prior research by Pu *et al.* [86]. We designed the rest of the questions for measuring the specific aspects that might be affected by our manipulation. While users are browsing recommendations in the explore page, we prompt users through a banner or pop-up; we started with the banner format and later turned to the pop-up format because the response rate of a banner is too low to collect enough survey feedback. Each user is prompted twice at maximum with the first and the second showing respectively three minutes or a week after a user joins the experiment (and if they show up browsing the explore page). We leave the survey link persistent on the page throughout the experiment so that users can give feedback anytime.

- *accuracy*: The recommendations match my tastes in movies.
- *diversity*: The recommendations have a diverse selection of movies.
- *novelty*: The recommendations help me discover interesting movies that I did not know.
- *temporary interest*: The recommendations reflect my recent interest in movies.
- *attractiveness*: I am interested in seeing or knowing more about the movies in the recommendations.
- *confusion*: I get disoriented sometimes by the change of the recommendations.
- *balance of recency*: The balance between new and old movies in the recommendations is appropriate to me.
- *balance of popularity*: The balance between popular and less popular movies in the recommendations is appropriate for me.
- *understandability*: I understand why the recommender is recommending the movies in my top-picks.

- *reactivity*: The recommendations change appropriately in reaction to what I do in MovieLens.
- *satisfaction*: Overall, I am satisfied by the recent recommendations from the recommender.

6.5 Results

Table 6.2: The coefficients (and standard errors in the parentheses) of the activity metrics predicting user overall satisfaction. These estimates come from an ordinal regression (cumulative link) model treating *satisfaction* response as ordinal values. Each observation is a user who has answered the *satisfaction* question for at least once.

Activity Metric	**Coefficient (vs. Satisfaction)**
log(#sessions)	0.524 (0.236)
log(#frontView)	0.0946 (0.213)
log(#exploreView)	-0.446 (0.152) *
frontPositive rate	0.522 (0.368)
frontNegative rate	-0.342 (0.302)
explorePositive rate	0.336 (0.245)
exploreNegative rate	-0.052 (0.162)

Table 6.3: The coefficients (and standard errors in the parentheses) of the perception metrics predicting user overall satisfaction. These estimates come from an ordinal regression (cumulative link) model treating *satisfaction* response as ordinal values while others as continuous values. Each observation is a user who has completed all the survey questions for at least once.

Perception	**Coefficient (vs. Satisfaction)**
accuracy	1.31 (0.282) *
diversity	0.595 (0.242)
novelty	-0.0294 (0.198)
temporary interest	0.847 (0.297) *
attractiveness	1.07 (0.241) *
confusion	-0.244 (0.167)
balance of recency	0.520 (0.209)
balance of popularity	-0.0165 (0.209)
understandability	0.255 (0.204)
reactivity	-0.198 (0.259)

The experiment was launched on Feb. 8, 2017 and the analysis was run on Aug.

Table 6.4: Means of both objective and subjective metrics for the six recommenders. Significant differences after correction are marked with *. Metrics that are not presented here do not show significant differences. The numbers in parentheses for #frontView and #exploreView are standard errors. These estimates come from two negative binomial regression models using MF-Rating as the baseline. The intervals for frontPositive rate, frontNegative rate, exploreNegative rate are 95% confidence intervals. These estimates come three mixed-effects logistic regression models using MF-Rating as the baseline (treating user ID as a random intercept effect). The means of *balance of recency* and *satisfaction* are calculated treating the survey responses as continuous values. The coefficients in the parentheses come from a ordinal regression (cumulative link) model using MF-Rating as the baseline (treating responses as ordinal values).

Recommender	#frontView	#exploreView	frontPositive rate	frontNegative rate	exploreNegative rate	balance of recency	satisfaction
MF-Rating	9.64 (0.733)	41.8 (4.47)	0.0740 (0.057, 0.0947)	0.0198 (0.0134, 0.0291)	0.175 (0.135, 0.224)	2.73 (N.A.)	2.89 (N.A.)
MF-Action	13.1 (1.00)*	72.6 (7.92)*	0.147 * (0.120, 0.180)	0.0521 * (0.0383, 0.0706)	0.330 * (0.272, 0.393)	3.20 (0.879)*	3.03 (0.253)
Bandit-Two	14.1 (1.15)*	48.4 (5.58)	0.133 * (0.105, 0.166)	0.0485 * (0.0346, 0.0674)	0.308 * (0.247, 0.377)	3.32 (0.969)*	3.00 (0.129)
Bandit-Four	13.9 (1.04)*	63.8 (6.78)*	0.125 * (0.102, 0.153)	0.0471 * (0.0350, 0.0631)	0.324 * (0.268, 0.386)	3.04 (0.501)	2.78 (-0.170)
Bandit-State	9.63 (0.741)	41.4 (4.44)	0.140 * (0.113, 0.173)	0.0605 * (0.0445, 0.0818)	0.327 * (0.264, 0.396)	3.26 (0.969)*	3.17 (0.485)
Reinforce-State	10.5 (0.828)	35.2 (3.92)	0.101 (0.0789, 0.128)	0.0405 * (0.0286, 0.0571)	0.213 (0.164, 0.271)	3.13 (0.647)	2.62 (-0.436)

27, 2017. During this period, 1508 users joined the experiment for at least one month, which gives us around 250 users for each recommender condition. They generated 12,627 sessions and 279,632 activities, including 92,289 ratings (41,651 of which ¿= 4.0), 34,587 clicks, and 12,919 wishlist additions. The remaining measured activities are mostly front page and explore page views. In total, we collected 3887 survey responses which gives us around 60 responses for each question on each recommender. We only use the last response of a user on a question if the user has multiple responses on the same question (we found using all survey responses of each user gives us the same results). We rely on different types of regression modeling techniques (including their p-values and effect sizes) treating MF-Rating as the baseline to draw conclusions and to avoid excessive pairwise comparisons. In addition, we performed the Benjamini-Hochberg correction procedure [107] to control the False Discovery Rate of the analysis (effectively using around p¡=0.0053 as the significance level). Before doing the analysis, we confirmed that the user activities were not significantly different for users in different conditions in terms of our activity measurements during the one month before they joined the experiment. This is to make sure that the effects come from our recommender manipulation.

6.5.1 Measurements Interpretation

We first conduct analysis to correlate user activity metrics and perception metrics with the user's overall satisfaction to understand how different types of activities and different perception aspects of the recommender contribute to its success or failure.

Table 6.2 and 6.3 show the results of predicting user overall satisfaction with both objective activity metrics and subjective perception metrics. We can see that perceived accuracy and attractiveness (interest users to know more about the items) are the two most important factors predicting user satisfaction. Whether recommendations reflect users' temporary interest is positively predicting satisfaction but with a smaller effect size. Table 6.2 also shows that the amount of browsing (*#exploreView*) is negatively associated with user satisfaction. The data shows a trend that higher positive action rate corresponds to higher user satisfaction while high negative action rate corresponds to lower user satisfaction.

6.5.2 RQ1

What are the differences between recommender systems based on explicit vs. implicit user feedback data modeled with the classical matrix factorization algorithms?

The first two lines of Table 6.4 compare MF-Rating (based on explicit feedback data) with MF-Action (based on implicit feedback data) on user objective activity measurements. Our overall measure of user satisfaction was not signficantly different between the two recommenders, but there are several differences in user activities. Specifically, MF-Action significantly increases the front page view, explore page view, frontPositive rate, frontNegative rate and exploreNegative rate. We find that in terms of the *balance of recency*, MF-Action is perceived to be significantly better than MF-Rating.

We did not find significant difference for metrics of *#sessions* (mean: 4.9 per user; similar analytical model as *#frontView*), *explorePositive* rate (mean=0.658; similar analytical model as *exploreNegative* rate) and *optOut* rate (mean=0.185; a logistic regression model) after correction. We did not find significant differences on the other subjective measurements that we deployed after correction.

6.5.3 RQ2

Do multi-factor-blending recommendation algorithms lead to improved or changed user experience and if so, how?

As shown in the rest of the Table 6.4, there is no significant difference on user satisfaction, but we see potential useful differences on user experience and activities. Bandit-Two recommender is very similar to MF-Action (sharing the benefit of increased front page view engagement, front page positive action rate, the cost of increased front page negative action rate and explore page negative action rate and improved perception of the balance of item recency) except that it does not significantly increases the amount of explore page view. From this aspect, Bandit-Two gains some potential benefit in terms of user satisfaction over MF-Action because explore page view is negatively predicting user satisfaction as shown in Table 6.2.

Bandit-Four recommender is almost identical to MF-Action in terms of net effects on user activities although it has very different inputs and algorithms. In other words, the introduced two additional factors (recency and popularity) compared with Bandit-Two seem to only increase the amount of explore page browsing. Bandit-State does not significantly affect the amount of front page view and explore page view, but significantly increases users' rate of actions compared with MF-Rating, including not only front page positive action rate but also front page negative and explore page negative action rate. Comparing with all previous recommenders, it seems to gain some benefits over MF-Rating, MF-Action and Bandit-Four, but has some cost compared with Bandit-Two because it loses the benefit of increased front page view engagement.

We did not find significant benefits in terms of the perceived balance of popularity and recency for Bandit-Four (which explicitly models these two factors) compared with Bandit-Two, MF-Action and MF-Rating, with our current model design. On contrary, Bandit-Two and Bandit-State gains some benefit in the perceived balance of recency compared with MF-Rating.

Reinforce-State recommender only has a significant negative effect on the front page action rate compared with MF-Rating and loses more benefits compared with Bandit-State and others. This is also consistent with our subjective measurements, where Reinforce-State has a trend of hurting many perceptional aspects including perceived accuracy, novelty and attractiveness compared with MF-Rating.

6.6 Discussion

The above section empirically demonstrates the differences between recommender systems based on user explicit rating feedback and implicit action feedback. Depending on the goals of the system, different recommendation algorithms might be used.

If the goal is to increase engagement, then predicting implicit action is much more effective than predicting explicit ratings. However, if the goal of the system is to improve user satisfaction, we need to be cautious not to exclusively optimize for predicting implicit action, because this type of recommender does not seem to be as precise as a rating-based recommender, as reflected by the increased negative user engagement, *e.g.*, negative action rate and increased user browsing effort. This observation points to the future direction of exploring to learn from both positive and negative feedback with a particular focus on penalizing items with negative user feedback.

Blending the two types of recommenders by optimizing for online user interaction achieves the same level of user engagement as using implicit-action-based recommender alone, but it does not lead to more user browsing, which seems to achieve a trade-off between the goals of user engagement and satisfaction. We think this effect may generalize to other algorithms modeling explicit or implicit feedback data because it likely reflects the fundamental property of the two types of feedback signals instead of the specific algorithms (although more studies are necessary to confirm). It also suggests that if we truly want to capture user satisfaction, we might need to go beyond ratings or actions, *e.g.*, measuring how the system assists the user's decision making tasks and optimizing for it.

Lastly, if the goal of the system is to improve users' perceived balance of recency on the recommendations, blending the two types of recommenders helps with this goal as well.

This work also provides design implications for future work on blending multiple factors (*i.e.*, ensemble of multiple recommenders) and utilizing user return as a learning feedback signal for recommender systems. Based on our observation in this work, we think that

- a better control is necessary (*e.g.*, by introducing a broader set of observations on the context of the user return) in order to use user return to explain whether a

session of recommendations are good or bad.

- the state (transition) model and the Q function approximation model need to have enough power (*e.g.*, linear models may not be enough as used in this work) to capture both the actual user state (transition) and the complexity of Q value (a projection of the future value of a recommendation). Recently, Wu *et al.* [134] proposed to use recurrent neural networks for the state transition model and demonstrated some benefits, which we think is a promising direction to try.

- the balance of recency or popularity are inherently personalized and high-order (*e.g.*, quadratic) because different users prefer different kinds of balance [125] and hence going beyond linear models here might be necessary as well.

We studied several new perceptional metrics for recommendations different from prior literature [86]. These factors are especially prominent in a live interactive recommender system, *e.g.*, whether the recommendations reflect users' temporary interest (which might be only valid in the field), whether users are interested to know more about the recommendations and whether users are disoriented by the change of the recommendations (changing dynamics are common in live systems, *e.g.*, Youtube, Amazon etc.). We find interesting results that besides the perceived accuracy, whether recommendations can interest users to know more about them and reflect users' temporary preference are significant positive factors in predicting the overall user satisfaction. However, future work is necessary to study the psychological constructs of these question items and how they (together with the behavioral metrics) can be generalized to platforms that directly involve user consumption (which could be a stronger type of implicit feedback signals compared with the user interactions with movie information used in this study and might demonstrate significant differences in these metrics).

Chapter 7

A Generic Recommender Server

7.1 Introduction

Recommender systems have been widely applied in various domains, *e.g.*, e-commerce, proactive information retrieval, personalized search, online entertainment etc. The core problem of recommendation is selecting and presenting items from a usually large item space, for which many effective algorithms and interfaces have been designed. Software tools, libraries, systems also have been developed for building recommender systems in various applications. Recommender system research has reached a tipping point where user-centric and algorithmic research can be closely combined to improve the user experience of recommender systems. However, we need new software tools to better support this type of work. The key to provide support is to accelerate the process of going from modeling and experimenting in an offline setting to online environments where users are interacting with the design of the system in real-time. We designed and developed a server framework to fulfill this goal. This chapter focuses on answering the following questions: "what is going from offline to online for recommender systems, why we want to go from offline to online and how?"

What is going from offline to online in recommender systems? Offline research or work suggests that it does not involve a regularly running system in production used by people. They might rely on a snapshot of data collected from online systems. In this setting, algorithms designed are typically for batch processing, not for real-time responding to user interactions. Online environment however involves a constantly

running system that deals with user requests and interactions in real-time whenever a user visits. Two aspects of a research can go from offline to online: *design* and *evaluation*. In the design aspect, it means designing for the user-system interaction process, going beyond historical statistical assumptions, being explicit in modeling user state changes with the environment. In the evaluation aspect, it means having the goal of user experience in mind, evaluating interfaces, model and algorithm design in front of people, answering questions of how these manipulations affect people in both lower-level perception, higher-level cognition and more broadly user personal development and social welfare.

Why do we want to go from offline to online in recommender systems? The reasons of going from offline to online go into three perspectives. One is from the perspective of studying the theories of human psychology and behavior. Recommender systems are designed for people to use to accomplish certain goals or tasks. The research of recommender systems largely involves people using the system and how we design computational models or algorithms to describe and support these people's tasks. From this perspective, a recommender system is an artifact, stimuli or a way of manipulation through which we study people. In this case, we have to put our system in front of people for them to use.

Another perspective starts from the limitation of machine learning theories. Simply treating recommendation problems as statistical learning problems may not be the right approach because there are mis-alignments between modeling assumption and the reality. The environment is dynamic which constantly involves distribution shifting. Recommender systems make real-time decisions in dynamic uncertain environments. Algorithms designed to learn from dynamic environments needs to be tested in user-facing systems to gain better ecological validity.

Lastly, a recommender system at its core is a decision-making support tool or information navigational support tool. The ultimate goal is for user experience, satisfaction and adoption of the technology. In the long run, we can better understand how recommender systems are affecting people's life, *e.g.*, answer the question of whether the algorithm or design is improving or hurting the well-being of people.

How do we go from from offline to online in recommender systems? Going from offline to online has benefits but also challenges. For example, when we think about

designing and evaluating a design in front of people, there are potentially two apparent obstacles. One obstacle is the access to real users. Another one is the cost of system implementation. In this work, we propose a generic recommender server framework to reduce the cost of implementation by extracting out the common components of recommender systems. Recommender system researchers or designers can now focus on one part and then re-use other parts. The barrier of access to users can be potentially overcome by designing user studies and recruiting people to try out new techniques (*e.g.*, from crowd-sourcing platforms, which however has limitation of generalization because the participants might not be real users of the system). We also envision that this server can become a open service that gathers and allocates application users as resources. What we hope to achieve in the long-run is a recommender system research platform where researchers and practitioners can easily plug-in their own design and get evaluation out of it from various domains of applications if these applications are hosted in the service. This work of designing a generic server is a necessary step in moving towards that direction

7.2 Related Work

There are many softwares that have been built to support building recommender systems. We categorize these softwares into four categories: *command-line tools, libraries with programming APIs, (distributed) systems* and *servers (or services)*, as shown in Table 7.1.

Some softwares implement specific models or algorithms, *e.g.*, Apex SVDFeature [21], SLIMx [8], MyMediaLite [135], LibFM [110], LibSVM [136] etc. which we categorize as command-line tools.

Some softwares provide with implementations for multiple models or algorithms and provide their capabilities through programming language APIs, *e.g.*, Lenskit for collaborative filtering based algorithms [89], scikit-learn [90] and SparkML [91] for various kinds of machine learning models. These also include many python, Java or R packages for building recommender systems. Researchers have developed general libraries to build models with complicated structures, *e.g.*, the recently popular platform TensorFlow [92]. Stan [93] also enables flexible specification of model structure and was

developed earlier than TensorFlow although it is more oriented towards offline data analysis.

Some softwares implemented generic modeling techniques or algorithms to run in clusters of machines. It could be specific models, *e.g.*, *xgboost* [108], *difacto* [137] etc. It could be general computation operators, *e.g.*, TensorFlow [92]. We categorize these softwares as *(distributed) systems*. *Prediction.io* [94] is a quite different distributed system because it is service-oriented. It has production serving and maintaining components and integrates a variety of models and algorithms through SparkML [91]. We categorize it as a predictive or recommendation server if it is used in recommender systems.

The generic server designed in this work is novel because of the following reasons (particularly compared with *Prediction.io*).

- Recommendation is different from machine learning, particularly retrieving and ranking candidates are conceptually different, additional functionalities to provide than building predictive models.
- In our server design, there is no hard separation between the event logging and the serving of predictions or recommendations in the architecture level, which is *Prediction.io*' design. This is because data processing especially feature extraction processes (described in details later) can be determined by specific recommendation models in the engine. This design enables the following two key benefits.
  - The online and offline loop is closed and there is not a hard separation between offline model building and online model serving, which helps achieve the goal of easily going from offline to online.
  - Real-world applications of recommender systems require a data processing and feature extraction framework that can flexibly take in different types of data sources and utilize them in the model. Our design better addresses this requirement as elaborated in details later.

In the following section, I describe in details the design goals, principles and specific implementations of the generic server, which we open sourced on GitHub [1] and name as *Samantha*.

[1] https://github.com/grouplens/samantha

Table 7.1: A high-level comparison of the available softwares for building recommender systems.

Software	General Domain	Software Type
Lenskit	recommendation	general tool/library
MyMediaLite	recommendation	general tool
SLIMx	recommendation	specific tool
libsvm **libFM** **Apex SVDFeature**	machine learning	specific tool
scikit-learn **Weka**	machine learning	general library
dmlc xgboost **dmlc difacto**	machine learning	distributed system
dmlc mxnet **theano** **caffee** **tensorflow**	deep(architectural/graphical) machine learning	general library distributed system
SparkML (Mahout)	machine learning	general library distributed system
Prediction.io	machine learning	server distributed system
Samantha	recommendation machine learning	general library server distributed system

7.3 The Generic Server Design

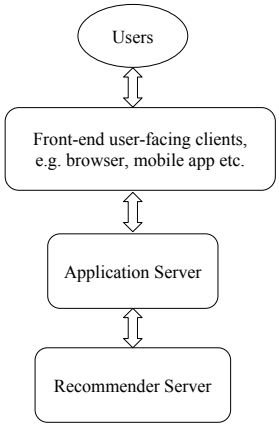

Figure 7.1: The environment that the generic recommender server is designed for.

Samantha, as a generic recommender server, is designed with the environment illustrated in Figure 7.1 in mind. It has four major parties: users, front-end user-facing clients, application servers and recommender servers. The recommender server only communicates with an application server, but not with the user-facing clients because of security reasons. Typically they communicate with each other through the HTTP

(HTTPS) protocol, *i.e.*, a recommender server is a web server similar to an application server, but specially focused on recommendation tasks. Going from offline to online means that the recommender server shipped with recommendation models and algorithms needs to respond to real-time requests and interaction feedbacks from users.

To enable this real-time recommendation serving capability, the following functionalities are necessary (this is what it takes to go from offline to online in summary for a recommender system):

- *data management*, including data storage and processing pipeline in response to user interactions or application content management
- *model management*, including online updating, building, loading, dumping and serving of the models
- *standard models and algorithms*, *e.g.*, collaborative filtering algorithms, machine learning regression or classification techniques
- *experimenting support*, *e.g.*, A/B testing, random or hashed, persistent assignment of users into different experimental or control conditions
- *real-time feedback loops*, for online machine learning and evaluation
- *extensibility for new model and algorithm design*, *e.g.*, the server can provide with parameter abstraction for model designers to freely design new computational algorithms on top of parameters without worry about where these parameters are stored and how to scale them up to large models; the server can also support parameter estimation by providing general optimization techniques or classic solvers, *e.g.*, stochastic gradient descent, to optimize for flexible objectives that come out from the model design process
- *flexible model dependency*, *e.g.*, model ensemble through boosting, bagging or stacking (*i.e.*, multiple levels of model dependency)
- *compatibility with other state-of-the-art systems*, *i.e.*, enable plugging in other implementations of recommendation or machine learning algorithms, general libraries

In this section, I describe how Samantha is designed to support all of these functionalities in a reusable, extensible and potentially scalable way so that researchers can focus on the most relevant part but still have a fully-functional system to go from offline to the online environment.

7.3.1 Recommender Components and Extensibility

In the server Samantha, *Recommender Engine* is referred to as a complete specification of how a recommender works. It supports multiple recommender engines, which is designed to scope applications. They are functionally similar to running multiple recommender servers. A recommender engine consists of eight types of components each of which can have multiple ones: *indexer*, *retriever*, *predictor*, *ranker*, *recommender*, *router*, *evaluator*, *scheduler*.

indexer. This type of component deals with data indexing in real-time when receiving data from applications. Since it knows how the data is stored, it is also responsible for outputting data from the behind storage for further batched data processing. By default, Samantha provides with multiple types of indexer implementations corresponding to different data storage back-end systems. Usually, one indexer is configured for one data type, similar to a database table, although one indexer type is general enough to take in any data types with different data fields. One can use one indexer for each data type the application sends in because it eases the integration between the recommender server and the application server. This will be elaborated in details later.

retriever. This type of component is responsible for retrieving candidates for a recommender engine. Recommender systems research typically focus on predictive model and algorithm innovation ignoring the problem of retrieving in an actual recommender system. Partially, it's because there are not many online experiments reported in details in academic and industrial research as to go into this aspect. However, when item space is big and especially when the predictive model involves complicated computation, an initial candidate generation process is a necessary component for a recommender engine. Retriever can be simple and straightforward, *e.g.*, retrieving all of the items or retrieving the top popular items. It can be complicated and critical to have good recommendations, too. For example, we can build simple machine learning models in a retriever to score all items and pick the top to generate candidates. Another possible approach is

to build fast associative models as an item-based k-nearest neighbor algorithm does, in which candidates become those most similar items to a user's previously liked items. We can also blend multiple retrievers with different priorities, *e.g.*, first using any results produced by a personalized retriever and resort to non-personalized one when necessary.

predictor. This type of component is the core part of operationalizing machine learning theories, roughly falling into the supervised learning domain. This is mainly a wrapper for a machine learning model implementation, which is essential for enabling Samantha to integrate with other machine learning libraries.

ranker. This type of component takes in an initial candidate item set and rank them based on certain criterion which could just be the output score of a predictor component.

recommender. In Samantha, a standard recommender is just the combination of a retriever and ranker, *i.e.*, a retriever retrieves initial candidates from the storage or an initial model and feeds them into ranker to generate the ranked list.

evaluator. There are two standard types of evaluators. Prediction evaluator provides the ability to compute prediction metrics for any predictor component. Recommendation evaluator evaluates any recommender component by computing top N recommendation metrics.

router. This type of component implements how to find the right recommender or predictor for a request, which is the foundation of A/B testing or between-subjects field evaluation framework.

scheduler. This type of component supports calling model management interfaces regularly with a predefined schedule, *e.g.*, for re-training a machine learning model.

Extending the capabilities of the generic server involves providing specific type of implementation for any type of recommender component that is relevant. For example, if the research is about designing new types of predictive models, it then involves implementing a predictor type after which the specification of the predictors in a recommender engine can be replaced with the new implementation but reuse the default available types of other components.

7.3.2 Server Interface, Architecture and Scalability

Samantha can be summarized in simple words as a HTTP server embedded with a recommendation framework. Before describing its interfaces and architecture, we first introduce the design of the Data Access Object (DAO) interface. Whenever the server interacts with outside data sources, it goes through a DAO implementation. All the implementations of the DAO interface summarizes the server's understanding on the possible data formats that the server can take in. When a request wants to tell the server to use a piece of data, it needs to tell the server which type of DAO is being used and how to construct that DAO with additional parameters passed in in the request.

Samantha's capabilities are provided to applications through three types of HTTP interfaces: data indexing, model management, recommendation (prediction) serving. This is the very front boundary of the server, through which the application server interacts with. Figure 7.2 illustrates the processing flow of Samantha receiving different types of requests.

Model management with parameters of component type, component name, model name, model operation. When model management requests come, Samantha first recognizes which type of component it is, *e.g.*, retriever or predictor and then finds the specific component (there could be multiple components with the same type, *e.g.*, multiple predictors). After this, the identified component will be constructed and do the actual work of managing its models. The request will also pass in DAO information if the model management task involves reading and processing a piece of data, *e.g.*, training a machine learning model with a given data source.

Data indexing with the parameters of indexer name, DAO information. If data are sent in for indexing, the server finds the specific indexer based on the indicated name in the request and ask the indexer to index data into the back-end storage system. Indexers support subscribers which means the application can ask to pass those data to other components at the same time in addition to being indexed so that other components can update themselves in real-time, *e.g.*, updating a machine learning model in a predictor according to an online optimization algorithm.

Recommendation and prediction serving with the parameters of user identification, context information. If the application server is requesting recommendations or predictions, Samantha first asks a router to identify a recommender or predictor. Then the

identified recommender or predictor is responsible for generating recommendation or prediction results. Before returning results, a wrapping process writes relevant information on the working recommender or predictor into the response in order to let the application know who generates the results (together with logging by the application, this enables analysis and comparison among different predictors or recommenders).

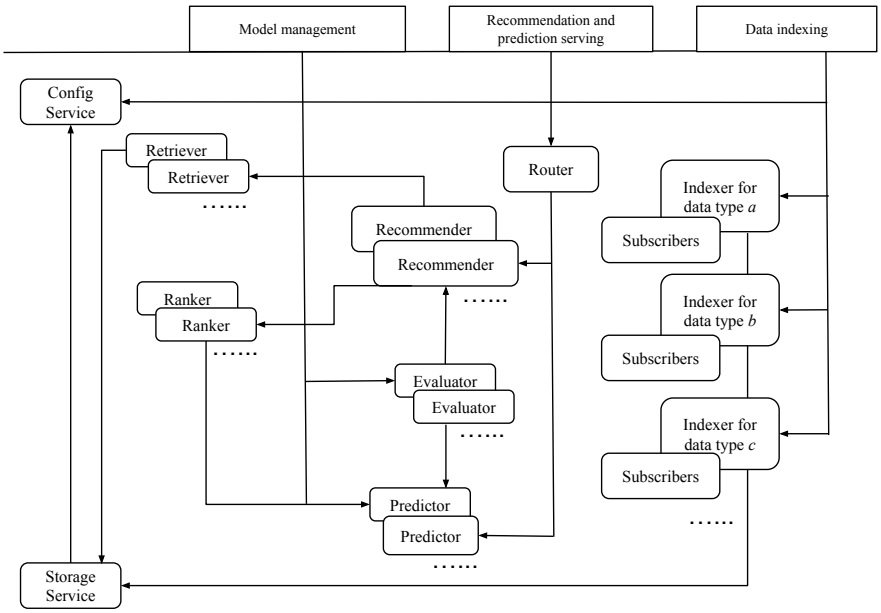

Figure 7.2: Samantha Data and Request Processing Flows. The directions of the arrows represent data flow and component dependencies.

Data processing pipeline

The data processing pipeline in Samantha consists of two concepts: *data expanders* and *feature extractors*. Data expanders are motivated by the observation that a complex model building requires data that goes beyond what one data type can offer or the raw content, interaction data sent in by the application requires future processing before being used for building models. Feature extractors are motivated by the observation that machine learning models require design matrix in which each data point is represented as a numerical vector containing the values of variables to be modeled, each of which is associated with certain parameters to estimate (*e.g.*, simple scalar coefficient or vector

embedding) depending the model design.

There are four common types of expanders: data joining, predictor based expander, filtering, (grouping and then) merging. We illustrate these expanders with an example. Imagine an application sends in a data point which is a tuple of user ID, item ID and rating. In order to build a potentially complex rating prediction model, we need to expand this data point according to how the rating prediction model is designed. For example, this model might relies on another model that tries to estimate how much the user likes tags of the item. We first need an expander that can communicate with the data storage service (*e.g.*, a relational database) for the tags of the item. This is a generic data expander because it is performing key-based query in a data service and join the search results with the data point. The next expander we need is a predictor based expander that takes in this data point and expands it with another model's predictions (assuming the user-tag preference model has been built in a predictor). If we want to exclude ratings that are out of the range [0.5, 5.0] because those might be invalid ratings or ratings designed for other purposes by the application, we need another filtering expander that filters the data point according to the criteria. If the rating prediction model actually needs all the user's rating history (*e.g.*, SVD++ [15]), having access to such individual ratings after querying the data storage service, we need a merging expander that combines all the previously rated items of the user into one data field and join it with the current data point. Grouping expanders might be needed when we are training a SVD++ prediction model (through model management requests) while the rating data provided by the DAO are individual ratings not grouped by users yet.

Even if we have all data fields available in the data point, a statistical model can not use it yet because it needs a numerical vector representation. This requires mapping a data field into an index space with which a corresponding parameter space is created. For example, user ID needs to be converted to an index that refers to its bias parameter in a vector parameter space or its latent factor parameters in a matrix parameter space for a standard matrix factorization model [18]. This is exactly what feature extractors are designed for. Each feature extractor takes in a data point and convert the relevant data field to be an index and a value (named collectively as a *feature*). For categorical

fields like user IDs, the value here is usually one, but for numerical variables (*e.g.*, predicted tag preference), the value here is the predicted preference score. After processing through a list of feature extractors, we have a numerical vector representation of a data point which can further be used by statistical models now.

Feature extractors rely on the capability of converting any string into the index of a densely organized variable or parameter space, which is referred as *index space*.

Why is this scalable?

The scalability of the server can be explained in three aspects. First, model size can go beyond what a single machine can hold utilizing distributed model training through the parameter server paradigm. Second, data sets can go beyond what a single machine supports and utilizing distributed storage system. Third, the serving of recommendation and predictions can be duplicated across multiple server clusters responding to potentially a large number of requests at the same time.

7.3.3 Using the Server

To use the server framework, what researchers and practitioners need to do now become

1. configure a recommender engine (explained below) to specify how each part should run

2. if the currently implemented state-of-the-art models or algorithms can not be used or the research itself is about new models or algorithms, design and develop new models or algorithms following the server framework

3. send in data generated by application content management or users interacting with the application front-end clients

4. ask for recommendations or predictions in real-time

A minimum recommender engine requires specifications on what data will be stored and how they are stored when data comes in in real-time, how an initial set of candidates will be retrieved and how to rank the candidates based on the prediction of a user

preference model. Assuming a standard matrix factorization model predicting user-item ratings, we illustrate below how to set up an engine in the recommender server to respond to application users in real-time.

indexers: One indexer that supports indexing user-item rating data into the server

- name: UserItemRatingIndexer
- implementation: database based indexer (*e.g.*, MySQL database based indexer)
- data fields: user ID, item ID, rating, time-stamp

retrievers: One retriever that retrieves all available items in the database store of the item data (note that the retrieved item list needs an expander to set the current user ID in the request)

- name: AllAvailableItemsRetriever
- implementation: database based retriever (*e.g.*, MySQL database based retriever)
- table: ItemData
- retrieved fields: item ID
- expanders: set the user ID data field in the request into the retrieved list of item data

predictors: One predictor that builds and maintains a matrix factorization model [18] with user, item bias terms and user, item latent factors.

- name: MatrixFactorizationRatingPredictor
- implementation: matrix factorization model based predictor
- feature extractors:
  - user bias extractor: convert user ID into the user index in the bias parameter space named as UserBias
  - item bias extractor: convert item ID into the item index in the bias parameter space named as ItemBias

  - user factor extractor: convert user ID into the user index in the latent factor parameter space named as UserFactor
  - item factor extractor: convert item ID into the item index in the latent factor parameter space named as ItemFactor

- model loss: L2-norm loss (regression)
- learning method: stochastic gradient descent

rankers: One ranker that ranks based on the predicted rating of the above rating predictor

- name: RatingPredictorBasedRanker
- implementation: a predictor based ranker which first makes predictions on a list of retrieved candidates and rank the list according to the predictions
- predictor name: MatrixFactorizationRatingPredictor

recommenders: One recommender that retrieves a candidate list and then ranks it based on the above ranker

- name: StandardRecommender
- implementation: a recommender that first retrieves a list of candidates with a retriever and ranks the candidates with a ranker
- retriever name: AllAvailableItemsRetriever
- ranker name: RatingPredictorBasedRanker

schedulers: One scheduler that trains the rating prediction model every day by taking in data from the indexed user-item ratings (note that the indexer supports both indexing, *i.e.*, data in, and outputting, *i.e.*, data out). Without this scheduler, the application server can still regularly sends in model management requests (described above section) to update the model.

- name: TrainingRatingModelScheduler

- implementation: a scheduler that mimics a model management request
- schedule: every mid-night
- request context: all available data in the indexer UserItemRatingIndexer

7.4 Case Studies

In this section I describe three case studies that demonstrate a) this server framework can easily integrate with state-of-the-art machine learning libraries and systems; b) this server framework enables complex online experiments with multi-level model dependencies or many important factors to model.

7.4.1 Extension and Integration

Integrating the generic server with other libraries or systems involves implementing a recommender component. This is exactly the same as writing its own implementation of a recommender component within the server. For example, the server provides with the implementation of a type of predictor based on the modeling technique SVDFeature [21]. When the component involves complex learning models or algorithms, the server provides support to further simplify by defining interfaces for the new component to implement. The interfaces for any recommender component that involves complex statistical machine learning models are as follows:

- *LearningInstance*: A feature list (or numerical vector) representation of a data point
- *LearningData*: An iterable set of LearningInstance representing a data set
- *LearningModel*: A complete representation of a model being able to extract features for a data point to get LearningInstance representation and make predictions on a LearningInstance
- *LearningMethod*: Train or update a LearningModel with a LearningData

TensorFlow

TensorFlow [92] is one of the most widely used machine learning libraries and systems nowadays. The powerful capabilities of TensorFlow enable not only flexible incorporation of data available in real-world applications, but also the flexible design of the modeling structure. We show here that the server framework we designed can easily integrate TensorFlow so that any recommendation models that are designed in TensorFlow with any type of computational graphs can run in the generic server leveraging the data processing pipeline, model management interfaces in Samantha.

Correspondingly, the integration takes the following implementation following the server framework (note that this integration is agnostic to the computational graphs):

- LearningInstance: A dictionary of tensors (called feed dictionary in TensorFlow) where the key is the name of the tensor defined in the computational graph and the value is the tensor representation of a data point
- LearningModel: Extract features by utilizing feature extractor interfaces and convert the list of indices and values into a TensorFlow specific LearningInstance; make predictions by feeding in the dictionary of tensors and running a specified operation in the computational graph
- LearningMethod: Iterate over a LearningData that outputs TensorFlow specific LearningInstance, feed in the dictionary of tensors and run a specified model updating operation in the computational graph (*e.g.*, a minimizer over a loss in the graph)
- predictor: When responding to prediction requests, this TensorFlow specific predictor asks the TensorFlow specific LearningModel to make predictions on the input data points. When responding to model management requests, it delegates the server to interpret a specific DAO and asks the TensorFlow specific LearningMethod to update or train the model with the DAO based LearningData representation.

Imagine that we want to run a TensorFlow graph defining a SVD++ [34] model. For simplicity of illustration, we dropped the bias terms here which then gives the following

prediction function, where U, Q and V are user, implicit action and item latent factor matrix and I is the set of items rated by user u.

$$f(u, I, a) = (U_u + \frac{1}{\sqrt[2]{|I|}} \sum_{i \in I} Q_i)^T V_a \tag{7.1}$$

The simplest definition of the a TensorFlow graph would be taking in four tensors (ignoring the bias terms for simplicity of illustration): a tensor (ImplicitFactor) with items rated by a user (note these items need to be indices referring to a matrix parameter space which suggests that the item IDs in the data sources should go through the feature extracting process in the data processing pipeline mentioned above), a tensor with the user (UserFactor, similarly an index referring to a matrix parameter space), a tensor with the target item (ItemFactor, similarly an index) to make prediction on and a tensor (Rating) with the actual rating given by the user on the item. To move this TensorFlow graph from offline to online in the generic server, we replace the predictor in section 7.3.3 with the following (note that it assumes the input data point has a data field with all the rated items by the user which can be easily achieved through the data expanders in the data processing pipeline of Samantha).

predictors: one predictor that is based on the TensorFlow predictor type

- name: TensorFlowSVDPlusPlusRatingPredictor
- implementation: the above mentioned TensorFlow specific predictor type
- graph: a file path pointing to the definition file of the TensorFlow graph
- feature extractors:
  - user factor extractor: convert user ID into the user index in the user latent factor parameter space named as UserFactor
  - item factor extractor: convert item ID into the item index in the item latent factor parameter space named as ItemFactor
  - implicit action extractor: convert a list of item IDs into the the item indices in the implicit action latent factor parameter space named as ImplicitFactor
  - rating value extractor: output the rating value as it is named as Rating

xgboost

xgboost [108] is also one of the widely used machine learning systems in various domains including recommender systems, *e.g.*, GBDT has been used in Yahoo! News recommendation [138]. Similarly, we show that xgboost can be easily integrated into the server framework and accelerate the process of moving a xgboost-based recommendation system from offline to online.

- LearningInstance: A labeled feature map from the feature index to feature value as required by xgboost
- LearningModel: Extract features by utilizing feature extractor interfaces and convert the list of indices and values into a xgboost specific LearningInstance; make predictions by getting results from a xgboost Booster
- LearningMethod: Iterate over a LearningData that outputs xgboost specific LearningInstance to create a xgboost specific data iterator and ask the xgboost library to train the xgboost LearningModel with the data iterator.
- predictor: When responding to prediction requests, this xgboost specific predictor asks the xgboost specific LearningModel to make predictions on the input data points. When responding to model management requests, it delegates the server to interpret a specific DAO and asks the xgboost specific LearningMethod to train the model with the DAO based LearningData representation.

7.4.2 Online Recommender Blending

A challenge of running recommendation models and algorithms in an online setting where application users can interact with the system any time lies at the continuous management of data and models, and dealing with complex multi-level model dependency. Samantha has been used by MovieLens to provide personalized movie recommendations [139]. Here I describe how one of the recommenders was implemented in Samantha which is based on the techniques of matrix factorization and LinearUCB [35].

The recommender has two levels of model dependency. In the first level, there are two matrix factorization models predicting rating ($f_1(u, a)$ in Equation 7.2) and

action probability ($f_2(u, a)$) respectively (where action is binary representing whether a displayed movie was interacted by the user or not). Conceptually, these two models are estimating how much a user might like a movie (predicted rating) and how likely a user might interact with a movie (predicted action probability). In the second level, there is one LinearUCB model that combines the predicted rating and action probability of a user-item pair to maximize the online interactions from users as a reward function by estimating the best set of weights for the two predictions. The label value of $r^*(u, a)$ also takes the value of either zero (when a movie is displayed but there is no action) or one (when a movie is displayed and also acted upon by the user).

$$r^*(u, a) = \beta_1 \cdot f_1(u, a) + \beta_2 \cdot f_2(u, a) \tag{7.2}$$

With already implemented predictor types supporting the matrix factorization and LinearUCB models, moving this recommender from offline to online environment does not need any additional implementation by using the following specifications in the recommender engine (continuing the minimum recommender engine example in Section 7.3.3).

indexers: One indexer that supports indexing user-item action data into the server

- name: UserItemActionIndexer
- implementation: database based indexer (*e.g.*, MySQL database based indexer)
- data fields: user ID, item ID, action, time-stamp

predictors: One predictor that builds and maintains a matrix factorization model [18] to predict action probability.

- name: MatrixFactorizationActionPredictor
- implementation: matrix factorization model based predictor
- feature extractors: the same as MatrixFactorizationRatingPredictor
- model loss: logistic loss (binary classification)
- learning method: stochastic gradient descent

Another predictor that online estimates a LinearUCB model to dynamically combine rating and action prediction models.

- name: LinearUCBActionPredictor
- implementation: LinearUCB model based predictor
- data expanders:
  - a predictor based expander that expands with the predicted rating of MatrixFactorizationRatingPredictor
  - a predictor based expander that expands with the predicted action probability of MatrixFactorizationActionPredictor
- feature extractors:
  - predicted rating value extractor: convert the predicted rating into a representation with both the blending weight index in the LinearUCB model and the predicted value
  - predicted action probability value extractor: convert the predicted action probability into a representation with both the blending weight index in the LinearUCB model and the predicted value
- learning method: LinearUCB specific learning method

schedulers: One scheduler that trains the action prediction model every day by taking in data from the indexed user-item actions.

- name: TrainingActionModelScheduler
- implementation: a scheduler that mimics a model management request
- schedule: every mid-night
- request context: all available data in the indexer UserItemActionIndexer

7.4.3 System-Level Cold-Start

One goal of Samantha is to support researchers who are interested in answering questions about how a predictive system or recommendation system affect user experience or user tasks. In this case study, we demonstrate how to set up a recommender server, with the minimum amount of implementation but without losing the flexibility of incorporating important factors in the domain, to study a new system that recommends emails to a user of an email client based on the user's calendar schedule [140]. We assume application developers have access to emails in a user's "inbox" or "sent" folder and the user's calendar schedule in the upcoming week after getting permission from the user (*i.e.*, acting as a agent for the user). Imagine that the application is a plugin for the email client that can proactively recommend potentially useful emails to the user based on the user's next upcoming meeting.

One can imagine a condition where the application provides an interface that enables real-time feedback from users so that users can tell the system whether the displayed emails are useful to the user's upcoming meeting or not (if the users want to) and the system interactively evolves by learning from this feedback. Alternatively, the system can directly learns from implicit user interactions, *e.g.*, treating hovering on or clicking the recommended email as positive signals. A particular reason that we want the system to learn in real-time is that the system does not have any usefulness feedback (neither explicit usefulness judgments or implicit interaction). The specification below following the recommender server framework we designed illustrates that it only takes minimum customization in the feature extraction process to have a production-ready recommender system for the project. All we need is a predictor specification similarly based on the LinearUCB model (other predictor types can be used as long as the predictor support online model learning or updating). Note that it assumes the input data sent by the email client (proxied by a secured application server) has four data fields for each data point: people involved in the email and meeting, content of the email and the meeting, *i.e.*, the application is asking (in the phase of requesting predictions) and telling (in the phase of training the model) the recommender how much useful a list of emails might be to a meeting. The researcher might hypothesize the content similarity between the email and meeting and the overlap of the sets of people involved are potentially important factors.

predictors: One predictor that online estimates a LinearUCB model to dynamically combine multiple factors that might be important for predicting the usefulness of emails to a meeting.

- name: LinearUCBEmailMeetingUsefulnessPredictor
- implementation: LinearUCB model based predictor type
- feature extractors:
  - email-meeting content similarity extractor: compute the similarity between the content of the email and meeting, set the extracted feature to be the parameter index of this factor and the computed similarity value
  - email-meeting people overlap extractor: compute the overlap between the sets of people involved in the email and meeting, set the extracted feature to be the parameter index of this factor and the computed overlap value

If the research develops customized complex modeling technique for predicting the usefulness of an email to a meeting, *e.g.*, implementing natively in the server or designing computational graphs in TensorFlow, the process of having this new technique take effects in front of users (or recruited participants) to get user feedback or user-centric evaluation can still be accelerated by using some of the components of the recommender server.

7.5 Discussion

In this work, we propose that recommender system research and practice can benefit from going from offline to online environments because it better connects the user-centric research and offline algorithmic and modeling research in recommender systems. In the long run, it can help answer the important questions of how recommendation technologies affect user perception, experience, satisfaction and people's life. We demonstrate that we can support this type of work by designing generic server frameworks. We provide an example framework along with case studies to show that this framework can support real world applications and potentially fulfill the goal of accelerating going from offline to online.

We recognize that not all research can benefit from following this approach. For example, computational questions for specific recommendation algorithms might be well answered with offline data experiments. Similarly, questions regarding user factors that are independent of algorithmic manipulation can be well answered by designed experimental tasks for users to accomplish without involving field usage of a system. For either type of work, following this server framework might incur significant overhead because it introduces constraints or additional work (*e.g.*, working with and maintaining a server environment). Overall, this work makes contribution by supporting an important thread of research in recommender systems where complex modeling and algorithms are combined with user-centric design and evaluation.

Chapter 8

Conclusion

User interaction is present in all user interfaces including recommender systems. Understanding user factors in interactive recommender systems is important for achieving better user experience and overall user satisfaction. Many prior works in recommender systems consider recommendation as a content selection process and there is not much prior work focusing on studying user interaction. My thesis studies several factors while real users are interacting with online recommender systems and answers a series of questions regarding those factors.

Assuming modern recommender systems presenting grids of recommendations to users page by page, we first studied user visual attention, *i.e.*, asking whether users pay attention to the displayed recommendations. We conducted initial research on modeling and predicting gaze in recommender systems with a grid-based interface. We applied HMM in this setting and achieved significant accuracy improvement in predicting fixation probability. We also showed that incorporating eye tracking data from a small number of users into the model training significantly boosts accuracy compared with only using normally logged user browsing data, even though the eye-tracked users are different from testing users. We found that user gaze behavior follows an *F-pattern* rather than showing a *center effect* in a grid-based interface. In addition, we find that user gaze behavior is different in different usage modes which suggests that collecting and training on a specific task is better, especially for system designers who have knowledge about the main task that their users are engaged in.

Further, we studied the interpretation of user inaction going beyond user attention

identifying other possible categories of reasons for user inaction inspired by the theories of human behavioral decision making. In recommender systems literature, inaction or missing observations is usually treated as negative feedback [104, 72, 111]. We demonstrated through a field survey in a live movie recommender system that inaction is more complex than the assumption. For example, there is a high chance that when a user skips a recommendation because the users wants to explore the recommendation later or the user has decided to accept the recommendation, user inaction could actually be positive feedback. We designed and tested models to infer user inaction and found that user inaction inference is generally a hard task. We achieved significantly better classification accuracy than naively predicting with the majority class and hence demonstrated that it is possible to infer user inaction, especially for certain categories, *e.g.*, whether the user has consumed a recommendation before. Lastly, we demonstrated that the inaction model inferring different inaction reasons can help improve action prediction and recommendation timing.

Since we know that users may not pay attention to all of recommendations displayed on a page, we might be better off spreading the best recommendations across multiple pages. We conduct an online field experiment to test two approaches of manipulating top-N recommendations: cycling, *i.e.*, the reranking of items in the top-N recommendation list based on users past exposure to these items, and serpentining, *i.e.*, the reranking of top-N item list by mixing the best-predicted items into later recommendation requests. We find interesting tensions between opt-outs and activities, user perceived accuracy and freshness. Intra-session cycling might be a "love it or hate it" recommender property, because users in it have a higher opt-out rate, but also engage in more activities such as page views, ratings, clicks and wishlistings, especially for those who stay. Inter-session cycling and serpentining increase activity without significantly increasing opt-out rate. Users perceive more change and freshness on cycled recommendations and less accuracy, familiarity on both cycled and serpentined recommendations. Combining cycling and serpentining does not work as well as each individual manipulation.

User interaction reflects user engagement with recommender systems. Besides better understanding the user interaction process with a system, we can also treat user interaction as an objective to optimize for, *i.e.*, designing recommendation models to

maximize the probability of user interaction or engagement. In this work, we conducted a large-scale randomized, controlled between-subjects field experiment to study six recommenders built using machine learning techniques, ranging from supervised matrix factorization, contextual bandit learning to Q learning, to compare recommenders that optimize for being accurate in preference estimation with ones that optimize for being engaging. We found that maximizing the user action probability of recommendations engages users more than maximizing the predicted rating or preference of users on recommendations modeled with the classical matrix factorization algorithms. However, the effects are mixed in a way that not only positive engagement but also negative engagement are increased substantially, which gives us a caution from the user's perspective on targeting implicit action only because overall user satisfaction could be negatively affected. We show that blending signals in both explicit and implicit user feedback through an online interactive learning algorithm gain the benefits of engagement and mitigate one of the possible costs (*i.e.*, the increased browsing effort). We tested the approach of utilizing user return as a delayed feedback signal on recommendation quality through Q-learning. It does not improve user experience with our current design but provides with potentially helpful implications for future work on recommender systems applying reinforcement learning.

Finally, my research promotes that recommender system research and practice can benefit from going from offline to online environments because it better connects the user-centric research and offline algorithmic and modeling research in recommender systems. In the long run, it can help answer the important questions of how recommendation technologies affect user perception, experience, satisfaction and people's life. We demonstrate that we can support this type of work by designing generic server frameworks. We provide an example framework through an open-source project along with case studies to show that this framework can support real world applications and fulfill the goal of accelerating going from offline to online.

8.1 Contribution

In summary, following the user-centric approach, my thesis has made contributions on the field of recommender systems by answering the following series of questions regarding

the online user-system interaction process in recommender systems, i.e.

- Do users pay attention to all the displayed recommendations in a typical grid-based recommender interface? How well can we predict user attention on the grid of recommendations?
  - Users on average only pay attention to fewer than half of the recommendations presented in the recommender interface. We can predict user attention on a grid of recommendations fairly accurately especially when boosted with eye tracking data from an initial set of users.
- How to interpret user inaction? how well can we infer the categories of user inaction reasons? How to utilize the inferred category of user inaction?
  - We summarized seven categories of reasons for user inaction which are *preference*, *context*, *attention*, *information*, *competition*, *decision* and *(re-)consumption*. Inferring the user inaction reasons is a hard tasks but we can achieve substantially better-than-random accuracy (47% for a 7-class classification task) and therefore we are able to utilize this inaction model to improve action prediction and potentially recommendation timing.
- Should we always present the top-N recommendations from best to worst? How does cycling and serpentining top-N item lists affect user experience?
  - We may want to delay our best recommendations into later page views to engage users and better utilize user attention when browsing recommendations. However, there is a trade-off between engagement and efficiency. For some users who want to see the best recommendations right away, delaying the presentation of best recommendations might negatively affect user perception on the usefulness of the recommender.
- How does treating user interaction or engagement as the objective to optimize for in recommender systems affect user experience, compared with optimizing for the traditional accuracy of user preference estimation? How to better optimize for user interaction or engagement?

  - A recommender maximizing the action probability of recommendations can better engage users than a recommender maximizing the preference of users on the recommendations. However, we need to be careful in targeting action only because this type of feedback data is noisier than user's explicit preference feedback and hence could substantially increase user negative engagement as well. Interactively blending the two recommenders through online learning algorithms can help gain the benefits of positive user engagement and mitigate some of the negative costs, *e.g.*, increased browsing effort from users.

- With user experience as the ultimate system goal, what type of software tools are possibly needed to accelerate the study of user-system interaction in recommender systems, especially for those based on complex learning algorithms?

  - We propose a generic recommender server framework that provides with a complete set of tools for all the functionalities that a recommendation algorithm needs to go from offline setting to online environments where real-time response to user recommendation requests is demanded. We demonstrate that this server framework is extensible (*e.g.*, it can easily integrate with state-of-the-art machine learning libraries and systems), flexible (*e.g.*, it enables complex model dependency and supports new types of research work based on online interactive learning) and potentially scalable.

8.2 Future Work

We envision two kinds of extensions to HMMs used in the work predicting gaze patterns of users on grids of recommendations. First, it is possible to consider individual-level, in addition to current global modeling if a user has enough page views. Second, the fixation duration on a position is modeled implicitly through state self-transitioning, which essentially assumes that the duration follows a geometric distribution [141]. This assumption may not be valid especially when more factors are introduced such as preferences to explain fixation duration. Hidden Semi-Markov Models (HSMM) have been proposed to account for it and successfully applied in speech recognition [142]. Applying

HSMM in our setting is a promising future direction.

An important piece of future work on top of the user inaction interpretation is to test the effects of this inaction model on user experience through designing and deploying field experiments. This can help answer the question of whether this model may improve recommendation freshness without hurting accuracy or ease the confusion of recommender systems that are based on dynamic algorithms learning from user action feedback neglecting user inaction.

The results of cycling and serpentining overall suggest that these two approaches of manipulating the top-N list have certain attractive properties, but future work is needed to design and test approaches that can increase freshness and even novelty without compromising accuracy, because we show that all of these aspects contribute positively to user-perceived usefulness and overall satisfaction. We also consider it necessary to conduct more detailed studies by inviting users into the lab, recording them using the system, and interviewing them in order to fully understand the effects. In addition to the presented results, we also find users using the recommender less often but reporting higher perceived accuracy. It suggests that an ideal recommender should be there to assist but not stand in the way to consumption, because most recommenders only provide access to the actual service being consumed. As suggested by Knijnenburg *et al.* [83] and McNee *et al.* [81], more future research on evaluating recommender systems from both system and user perspectives is desired.

Regarding the generic recommender server, as possible future work, we plan to design visualization tools to ease the process of creating and customizing recommendation engines so that it can support creating experimental platforms where cutting-edge academic research results easily go into applications to be evaluated and adopted. We might also explore supporting security and access control to expose the server to the outside web instead of hiding behind an application server to target being a open recommendation service. Another promising future work is to provide support for front-end presentation and logging of an application, *i.e.*, the server returns not only a list of results but also a formatting instruction indicating how the results should be displayed to users. With this support, we can enable embedding standard user-centric evaluation instruments (*e.g.*, carefully designed survey questions), gaining a better understanding of the relationships between easy-to-compute offline metrics and hard-to-measure user

experience.

8.3 Implication

My research on studying the online interaction process of users in recommender systems has implications on two potentially significant directions in the field.

First, focusing on going from offline to online enables establishing cross-domain experimental platforms that support fast user-centric field evaluation of recommendation techniques to gain a better understanding (ideally developing integrative theories on) how recommendation techniques impact user experience and people's life. For example, we need build systematic understanding on how algorithmic manipulation affects user perception, cognition and development. In this booming era of artificial intelligence and its increasingly large influence on our societies, this thread of work can be critical and pressing.

Second, building recommender systems that are mindful of user interaction process can go beyond creating delightful user experience. It leads to research on systems that promote user interaction and benefit users from their deep interaction with the system. For example, a recommender system that interactively presents online learners with learning materials based on the user's mental state can better motivate learners to learn and encourage deep learning processes to happen. These potential advancements can play a significant role in developing the next-generation personalized learning environments in both traditional and online education.

References

[1] Shuo Chang, F Maxwell Harper, and Loren Terveen. Using groups of items for preference elicitation in recommender systems. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 1258–1269. ACM, 2015.

[2] Dan Cosley, Shyong K Lam, Istvan Albert, Joseph A Konstan, and John Riedl. Is seeing believing?: how recommender system interfaces affect users' opinions. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 585–592. ACM, 2003.

[3] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. *ACM TOIS*, 22(1):5–53, 2004.

[4] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

[5] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[6] Joseph A Konstan and Ekstrand Michael D. Introduction to recommender systems. Coursera.org, 2017.

[7] Simon Funk. Netflix update: Try this at home, 2006.

[8] Xia Ning and George Karypis. Slim: Sparse linear methods for top-n recommender systems. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 497–506. IEEE, 2011.

[9] Evangelia Christakopoulou and George Karypis. Local item-item models for top-n recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 67–74. ACM, 2016.

[10] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, pages 191–226. Springer, 2015.

[11] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 79–86. ACM, 2010.

[12] Gediminas Adomavicius and YoungOk Kwon. Multi-criteria recommender systems. In *Recommender systems handbook*, pages 847–880. Springer, 2015.

[13] Mario Rodriguez, Christian Posse, and Ethan Zhang. Multiple objective optimization in recommender systems. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 11–18. ACM, 2012.

[14] Chhavi Rana and Sanjay Kumar Jain. A study of the dynamic features of recommender systems. *Artificial Intelligence Review*, pages 1–13, 2015.

[15] Yehuda Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97, 2010.

[16] Wenxing Hong, Lei Li, and Tao Li. Product recommendation with temporal dynamics. *Expert systems with applications*, 39(16):12398–12406, 2012.

[17] Linas Baltrunas and Xavier Amatriain. Towards time-dependant recommendation based on implicit feedback. In *Workshop on context-aware recommender systems (CARS09)*, 2009.

[18] Yehuda Koren, Robert Bell, Chris Volinsky, et al. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[19] Léon Bottou and Yann LeCun. Large scale online learning. In *NIPS*, volume 30, page 77, 2003.

[20] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, DTIC Document, 2000.

[21] Tianqi Chen, Weinan Zhang, Qiuxia Lu, Kailong Chen, Zhao Zheng, and Yong Yu. Svdfeature: a toolkit for feature-based collaborative filtering. *Journal of Machine Learning Research*, 13(Dec):3619–3622, 2012.

[22] Josef Bauer and Alexandros Nanopoulos. A framework for matrix factorization based on general distributions. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 249–256. ACM, 2014.

[23] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[24] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[25] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[26] Yue Wang, Dawei Yin, Luo Jie, Pengyuan Wang, Makoto Yamada, Yi Chang, and Qiaozhu Mei. Beyond ranking: Optimizing whole-page presentation. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 103–112. ACM, 2016.

[27] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19, 2016.

[28] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA, 2007.

[29] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to recommender systems handbook*. Springer, 2011.

[30] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296. ACM, 2010.

[31] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010.

[32] Dirk Bollen, Mark Graus, and Martijn C Willemsen. Remembering the stars?: effect of time on preference retrieval from memory. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 217–220. ACM, 2012.

[33] Christopher JC Burges, Robert Ragno, and Quoc Viet Le. Learning to rank with nonsmooth cost functions. In *NIPS*, volume 6, pages 193–200, 2006.

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

[35] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670. ACM, 2010.

[36] Guy Shani, David Heckerman, and Ronen I Brafman. An mdp-based recommender system. *Journal of Machine Learning Research*, 6(Sep):1265–1295, 2005.

[37] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT Press Cambridge, 1998.

[38] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. Recurrent recommender networks. *Training*, 10(2):10–1.

[39] Nan Jiang and Lihong Li. Doubly robust off-policy value evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 652–661, 2016.

[40] Georgios Theocharous, Philip S Thomas, and Mohammad Ghavamzadeh. Personalized ad recommendation systems for life-time value optimization with guarantees. In *IJCAI*, pages 1806–1812, 2015.

[41] Timothy J Buschman and Earl K Miller. Top-down versus bottom-up control of attention in the prefrontal and posterior parietal cortices. *science*, 315(5820):1860–1862, 2007.

[42] Claudia Roda. Human attention and its implications for human-computer interaction. *Human Attention in Digital Environments*, 1:11–62, 2011.

[43] John M Findlay and Iain D Gilchrist. *Active vision: The psychology of looking and seeing*. Number 37. Oxford University Press, 2003.

[44] David Marr and Tomaso Poggio. A computational theory of human stereo vision. *Proceedings of the Royal Society of London B: Biological Sciences*, 204(1156):301–328, 1979.

[45] Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 20(11):1254–1259, 1998.

[46] Tilke Judd, Krista Ehinger, Frédo Durand, and Antonio Torralba. Learning to predict where humans look. In *Computer Vision, 2009 IEEE 12th international conference on*, pages 2106–2113. IEEE, 2009.

[47] Moran Cerf, Jonathan Harel, Wolfgang Einhäuser, and Christof Koch. Predicting human gaze using low-level saliency combined with face detection. In *Advances in neural information processing systems*, pages 241–248, 2008.

[48] Benjamin W Tatler, Mary M Hayhoe, Michael F Land, and Dana H Ballard. Eye guidance in natural vision: Reinterpreting salience. *Journal of vision*, 11(5):5–5, 2011.

[49] Benjamin W Tatler and Benjamin T Vincent. The prominence of behavioural biases in eye guidance. *Visual Cognition*, 17(6-7):1029–1054, 2009.

[50] Stephen R Ellis and Lawrence Stark. Statistical dependency in visual scanning. *Human factors*, 28(4):421–438, 1986.

[51] Selim S Hacisalihzade, Lawrence W Stark, and John S Allen. Visual perception and sequences of eye movement fixations: A stochastic modeling approach. *IEEE Transactions on systems, man, and cybernetics*, 22(3):474–481, 1992.

[52] John M Henderson, Svetlana V Shinkareva, Jing Wang, Steven G Luke, and Jenn Olejarczyk. Predicting cognitive state from eye movements. *PloS one*, 8(5):e64937, 2013.

[53] Amin Haji-Abolhassani and James J Clark. A computational model for task inference in visual search. *Journal of vision*, 13(3):29–29, 2013.

[54] Soussan Djamasbi, Marisa Siegel, and Tom Tullis. Visual hierarchy and viewing behavior: An eye tracking study. In *International Conference on Human-Computer Interaction*, pages 331–340. Springer, 2011.

[55] Benjamin W Tatler. The central fixation bias in scene viewing: Selecting an optimal viewing position independently of motor biases and image feature distributions. *Journal of vision*, 7(14):4–4, 2007.

[56] Laura A Granka, Thorsten Joachims, and Geri Gay. Eye-tracking analysis of user behavior in www search. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 478–479. ACM, 2004.

[57] Eugene Agichtein, Eric Brill, and Susan Dumais. Improving web search ranking by incorporating user behavior information. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 19–26. ACM, 2006.

[58] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 154–161. Acm, 2005.

[59] Ye Chen and Tak W Yan. Position-normalized click prediction in search advertising. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 795–803. ACM, 2012.

[60] Georges E Dupret and Benjamin Piwowarski. A user browsing model to predict search engine click data from past observations. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 331–338. ACM, 2008.

[61] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. An experimental comparison of click position-bias models. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 87–94. ACM, 2008.

[62] Ramakrishnan Srikant, Sugato Basu, Ni Wang, and Daryl Pregibon. User browsing models: relevance versus examination. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 223–232. ACM, 2010.

[63] Olivier Chapelle and Ya Zhang. A dynamic bayesian network click model for web search ranking. In *Proceedings of the 18th international conference on World wide web*, pages 1–10. ACM, 2009.

[64] Andrew T Duchowski, Nathan Cournia, and Hunter Murphy. Gaze-contingent displays: A review. *CyberPsychology & Behavior*, 7(6):621–634, 2004.

[65] Jeff Huang, Ryen White, and Georg Buscher. User see, user point: gaze and cursor alignment in web search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1341–1350. ACM, 2012.

[66] Qi Guo and Eugene Agichtein. Towards predicting web searcher gaze position from mouse movements. In *CHI'10 Extended Abstracts on Human Factors in Computing Systems*, pages 3601–3606. ACM, 2010.

[67] Georg Buscher, Ludger van Elst, and Andreas Dengel. Segment-level display time as implicit feedback: a comparison to eye tracking. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 67–74. ACM, 2009.

[68] Georg Buscher, Edward Cutrell, and Meredith Ringel Morris. What do you see when you're surfing?: using eye tracking to predict salient regions of web pages. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 21–30. ACM, 2009.

[69] Sylvain Castagnos, Nicolas Jones, and Pearl Pu. Eye-tracking product recommenders' usage. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 29–36. ACM, 2010.

[70] Songhua Xu, Hao Jiang, and Francis Lau. Personalized online document, image and video recommendation via commodity eye-tracking. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 83–90. ACM, 2008.

[71] Kai Puolamäki, Jarkko Salojärvi, Eerika Savia, Jaana Simola, and Samuel Kaski. Combining eye movements and collaborative filtering for proactive information retrieval. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 146–153. ACM, 2005.

[72] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272. Ieee, 2008.

[73] Katja Hofmann, Anne Schuth, Alejandro Bellogin, and Maarten De Rijke. Effects of position bias on click-based recommender evaluation. In *European Conference on Information Retrieval*, pages 624–630. Springer, 2014.

[74] Hillel J Einhorn and Robin M Hogarth. Behavioral decision theory: Processes of judgement and choice. *Annual review of psychology*, 32(1):53–88, 1981.

[75] Jerome R Busemeyer and James T Townsend. Decision field theory: A dynamic-cognitive approach to decision making in an uncertain environment. *Psychological review*, 100(3):432, 1993.

[76] Jerome R Busemeyer and Joseph G Johnson. Computational models of decision making. *Blackwell handbook of judgment and decision making*, pages 133–154, 2004.

[77] Marius Usher and James L McClelland. The time course of perceptual choice: the leaky, competing accumulator model. *Psychological review*, 108(3):550, 2001.

[78] Frank Y Guo and Keith J Holyoak. Understanding similarity in choice behavior: A connectionist model. In *Proceedings of the twenty-fourth annual conference of the cognitive science society*, pages 393–398, 2002.

[79] John S Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 43–52. Morgan Kaufmann Publishers Inc., 1998.

[80] Yue Shi, Martha Larson, and Alan Hanjalic. List-wise learning to rank with matrix factorization for collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 269–272. ACM, 2010.

[81] 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.

[82] Joseph A Konstan and John Riedl. Recommender systems: from algorithms to user experience. *User modeling and user-adapted interaction*, 22(1):101–123, 2012.

[83] Bart P Knijnenburg, Martijn C Willemsen, Zeno Gantner, Hakan Soncu, and Chris Newell. Explaining the user experience of recommender systems. *User Modeling and User-Adapted Interaction*, 22(4-5):441–504, 2012.

[84] Martin Fishbein and Icek Ajzen. Belief, attitude, intention, and behavior: An introduction to theory and research. 1977.

[85] Sean M McNee, John Riedl, and Joseph A Konstan. Making recommendations better: an analytic model for human-recommender interaction. In *CHI'06 extended abstracts on Human factors in computing systems*, pages 1103–1108. ACM, 2006.

[86] Pearl Pu, Li Chen, and Rong Hu. A user-centric evaluation framework for recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 157–164. ACM, 2011.

[87] Viswanath Venkatesh, Michael G Morris, Gordon B Davis, and Fred D Davis. User acceptance of information technology: Toward a unified view. *MIS quarterly*, pages 425–478, 2003.

[88] Bo Xiao and Izak Benbasat. E-commerce product recommendation agents: use, characteristics, and impact. *MIS quarterly*, 31(1):137–209, 2007.

[89] Michael D Ekstrand, Michael Ludwig, Joseph A Konstan, and John T Riedl. Rethinking the recommender research ecosystem: reproducibility, openness, and lenskit. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 133–140. ACM, 2011.

[90] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[91] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

[92] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[93] Bob Carpenter, Andrew Gelman, Matt Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Michael A Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of Statistical Software*, 20, 2016.

[94] Simon Chan, Thomas Stone, Kit Pang Szeto, and Ka Hou Chan. Predictionio: a distributed machine learning server for practical software development. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2493–2496. ACM, 2013.

[95] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *CSCW*, 1994.

[96] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In *ICDM'08*, pages 502–511. IEEE, 2008.

[97] Mackenzie G Glaholt and Eyal M Reingold. Eye movement monitoring as a process tracing methodology in decision making research. *Journal of Neuroscience, Psychology, and Economics*, 4(2):125, 2011.

[98] Claudia Roda. 2 human attention and its implications for human–computer interaction. *Human Attention in Digital Environments*, 1:11–62, 2011.

[99] John M Findlay. Active vision: The psychology of looking and seeing. 2014.

[100] David Marr, Tomaso Poggio, Ellen C Hildreth, and W Eric L Grimson. *A computational theory of human stereo vision*. Springer, 1991.

[101] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[102] John Mullahy. Specification and testing of some modified count data models. *Journal of econometrics*, 33(3):341–365, 1986.

[103] Shuang-Hong Yang, Bo Long, Alexander J Smola, Hongyuan Zha, and Zhaohui Zheng. Collaborative competitive filtering: learning recommender using context of user choice. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 295–304. ACM, 2011.

[104] Pei Lee, Laks VS Lakshmanan, Mitul Tiwari, and Sam Shah. Modeling impression discounting in large-scale recommender systems. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1837–1846. ACM, 2014.

[105] Qian Zhao, Gediminas Adomavicius, F Maxwell Harper, Martijn Willemsen, and Joseph A Konstan. Toward better interactions in recommender systems: Cycling and serpentining approaches for top-n item lists. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, pages 1444–1453, 2017.

[106] Qian Zhao, Shuo Chang, F Maxwell Harper, and Joseph A Konstan. Gaze prediction for recommender systems. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 131–138. ACM, 2016.

[107] Yosef Hochberg and Yoav Benjamini. More powerful procedures for multiple significance testing. *Statistics in medicine*, 9(7):811–818, 1990.

[108] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, 2016.

[109] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[110] Steffen Rendle. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57, 2012.

[111] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

[112] Cai-Nicolas Ziegler, Sean M McNee, Joseph A Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, pages 22–32. ACM, 2005.

[113] Lex Van Velsen, Thea Van Der Geest, Rob Klaassen, and Michael Steehouder. User-centered evaluation of adaptive and adaptable systems: a literature review. *The knowledge engineering review*, 23(3):261–281, 2008.

[114] Nisheeth Srivastava and Paul Schrater. Learning what to want: context-sensitive preference learning. *PloS one*, 10(10):e0141129, 2015.

[115] Xin Luo, Yunni Xia, and Qingsheng Zhu. Incremental collaborative filtering recommender based on regularized matrix factorization. *Knowledge-Based Systems*, 27:271–280, 2012.

[116] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th annual international conference on machine learning*, pages 689–696. ACM, 2009.

[117] Tyler Lu, Dávid Pál, and Martin Pál. Contextual multi-armed bandits. In *Proceedings of the Thirteenth international conference on Artificial Intelligence and Statistics*, pages 485–492, 2010.

[118] Gerard J Tellis. Advertising exposure, loyalty, and brand purchase: A two-stage model of choice. *Journal of marketing research*, pages 134–144, 1988.

[119] Colin McDonald. *What is the short-term effect of advertising.* Mass. Marketing Science Institute, 1971.

[120] Richard E Petty and John T Cacioppo. The elaboration likelihood model of persuasion. In *Communication and persuasion*, pages 1–24. Springer, 1986.

[121] Komal Kapoor, Karthik Subbian, Jaideep Srivastava, and Paul Schrater. Just in time recommendations: Modeling the dynamics of boredom in activity streams. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pages 233–242. ACM, 2015.

[122] Pearl Pu, Li Chen, and Rong Hu. Evaluating recommender systems from the users perspective: survey of the state of the art. *User Modeling and User-Adapted Interaction*, 22(4-5):317–355, 2012.

[123] Vladimir Vapnik. Principles of risk minimization for learning theory. In *Advances in neural information processing systems*, pages 831–838, 1992.

[124] Xavier Amatriain and Justin Basilico. Netflix recommendations: beyond the 5 stars (part 1). *Netflix Tech Blog*, 6, 2012.

[125] F Maxwell Harper, Funing Xu, Harmanpreet Kaur, Kyle Condiff, Shuo Chang, and Loren Terveen. Putting users in control of their recommendations. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 3–10. ACM, 2015.

[126] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[127] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B Kantor. *Recommender systems handbook*. Springer, 2015.

[128] Charu C Aggarwal. *Recommender systems*. Springer, 2016.

[129] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[130] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[131] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[132] Shai Shalev-Shwartz and Yoram Singer. Online learning: Theory, algorithms, and applications. 2007.

[133] Nan Jiang and Lihong Li. Doubly robust off-policy value evaluation for reinforcement learning. *arXiv preprint arXiv:1511.03722*, 2015.

[134] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. Recurrent recommender networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 495–503. ACM, 2017.

[135] Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Mymedialite: a free recommender system library. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 305–308. ACM, 2011.

[136] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):27, 2011.

[137] Mu Li, Ziqi Liu, Alexander J Smola, and Yu-Xiang Wang. Difacto: Distributed factorization machines. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 377–386. ACM, 2016.

[138] Xing Yi, Liangjie Hong, Erheng Zhong, Nanthan Nan Liu, and Suju Rajan. Beyond clicks: dwell time for personalization. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 113–120. ACM, 2014.

[139] Qian Zhao, F Maxwell Harper, Gediminas Adomavicius, and Joseph A Konstan. Explicit or implicit feedback? engagement or satisfaction? a field experiment on machine learning based recommender systems. In *Proceedings of the 33rd ACM/SIGAPP Symposium On Applied Computing*. ACM, 2018.

[140] Qian Zhao, Bennett Paul N., Adam Fourney, Anne Loomis Thompson, Shane Williams, Adam D. Troy, and Susan T. Dumais. Calendar-aware proactive email recommendation. In *Proceedings of The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, 2018.

[141] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proc. IEEE*, 77(2):257–286, 1989.

[142] Jack D Ferguson. Variable duration models for speech. In *Proc. Symposium on the Application of HMMs to Text and Speech*, pages 143–179, 1980.