

USER-CENTRIC RECOMMENDATION SYSTEMS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Longqi Yang

August 2019

© 2019 Longqi Yang
ALL RIGHTS RESERVED

USER-CENTRIC RECOMMENDATION SYSTEMS

Longqi Yang, Ph.D.

Cornell University 2019

People's daily actions and decisions are increasingly shaped by recommendation systems (recommenders) that selectively suggest and present information items, from e-commerce and content platforms to education and wellness applications. However, existing systems are often optimized to promote commercial metrics, such as click-through rates and sales, while overlooking utility for individual users. As a result, recommendations can be narrow, skewed, homogeneous, and divergent from users' aspirations.

This thesis introduces **user-centric recommendation systems** that are built to optimize for individuals' benefit. These systems advance the state of the art of recommenders by addressing the bias and incompleteness of implicit feedback upon which existing systems rely, such as click, download, and share. Specifically, this thesis explores three research directions: (1) **Debiasing implicit feedback**. We leverage a Self-Normalized Inverse-Propensity-Scoring (SNIPS) technique to derive a debiased measure of recommendation performance. Our approach models and alleviates popularity bias and is shown to significantly reduce the Mean Absolute Error (MAE) of evaluating recommendation systems offline. (2) **Leveraging richer data sources to learn broader user preferences**. We develop an unsupervised learning algorithm to learn discriminative user representations from unstructured software usage traces. The learned representations significantly improve the performance of personalization systems for creative professionals, including creative content recommenders and user tag-

ging systems. (3) **Interactive preference learning addressing the limitations of passively collected offline data.** We build an interactive learning framework to learn users' food preferences from adaptive pairwise comparisons. This framework enables a recipe recommender that satisfies users' tastes and nutritional expectations. We also design an onboarding survey to empower an intention-informed podcast recommender. Through lab and field experiments, we demonstrate that these systems can promote healthier diets and aspiration-aligned content choices.

In addition to the aforementioned user-centric recommenders, this thesis also contributes an open source tool, named OpenRec, to tackle the challenges of model generalization and adaptation that arise in building heterogeneous recommendation systems. OpenRec provides modular interfaces so that monolithic algorithms can be readily decomposed or combined for diverse application scenarios. At the end of this thesis, we discuss future research to personalize pervasive intelligent systems for people and our society and to understand and mitigate the unintended consequences of personalization.

BIOGRAPHICAL SKETCH

Longqi Yang grew up in Suichang, China and received his B.Eng. in Information and Communication Engineering from Zhejiang University in 2014. Since then, he has pursued a Ph.D. in Computer Science at Cornell University under the supervision of Prof. Deborah Estrin. He conducted his thesis research as a part of the Connected Experiences (Cx) Lab and the Small Data Lab, both at Cornell Tech. His research has focused on personalization, recommendation systems, and machine learning for user behavior modeling. In this line of work, Longqi has made various types of contributions, including developing novel recommendation methods and algorithms using advanced machine learning and deep learning techniques, building deployable systems, and conducting lab and field experiments. His work has been deployed and adopted commercially and recognized at flagship industrial and academic conferences.

Dedicated to my family.

ACKNOWLEDGEMENTS

First and foremost, I have been incredibly lucky to have Deborah Estrin as my Ph.D. advisor. She has been, and will continue to be, my role model as a researcher, advisor, and colleague. What I have learned from her is far beyond how to communicate, conduct research, mentor students, and collaborate with others. Words are not enough to express my gratitude for her support, guidance, and trust over the past five years. I will demonstrate the spirit that I have inherited from her through my work in the years to come.

I have also been extremely fortunate to have Serge Belongie, Mor Naaman, Nicola Dell, and Claire Cardie as my thesis committee members. Working with them and being exposed to ideas and perspectives from their areas of expertise has played a central role in the development of my research interests and values. They have always been great supporters throughout my Ph.D. study.

In addition, this thesis would not have been possible without my excellent co-authors, including my advisors, collaborators, lab mates, and mentees: Deborah Estrin, Serge Belongie, Mor Naaman, Nicola Dell, Curtis Cole, Cheng-Kang Hsieh, Hongjian Yang, JP Pollak, Min Aung, Faisal Alquaddoomi, Mashfiqui Rabbi, Tanzeem Choudhury, Chen Fang, Hailin Jin, Matthew D. Hoffman, Hongyi Wen, Michael Sobolev, Yu Wang, Jenny Chen, Drew Dunne, Christina Tsangouri, Yin Cui, Yuan Xuan, Chenyang Wang, Eugene Bagdasaryan, Joshua Gruenstein, Tsung-Yi Lin, Honghao Wei, Diana Freed, Alex Wu, Judy Wu, and Fan Zhang. I would also like to thank many administrative staff members who made almost everything that I did as smooth as possible: Becky Stewart, Donna Rose, Jessie Taft, Juliana Kleist-Mendez, Devaneke Crumpler, Sarah Abdelnour, and Tamika Morales.

My Ph.D. experience has been made unique by Cornell Tech, where I am

grateful to be one of the first cohorts of Ph.D. students, including Andreas Veit, Kimberly Wilber, Yin Cui, Fan Zhang, Xiao Ma, Lei Shi, Yuhang Zhao, Yuhang Ma, and Fabian Okeke. Growing with the campus and being surrounded by people who are passionate about real-world impact is a truly once-in-a-lifetime opportunity. I especially value the chances that I have gotten to interact with people from other disciplines beyond computer science, which has always been a great source of inspiration.

Last but not least, my debts to my parents, Quanjun Yang and Mingfei Wang, are unbounded. They have provided unconditional support and love throughout my life and have always encouraged me to be brave to challenge myself and to make a positive impact on the world. I also want to sincerely thank my fiancée, Xiaoti Hu, who has always been by my side since we were together at college. Without her selfless support, trust and sacrifice, getting through the rough time of pursuing my Ph.D. would not have been possible.

The research included in this thesis was supported by the National Science Foundation under grant IIS-1700832 and by Yahoo Research (via the Connected Experiences Laboratory at Cornell Tech). The work was further supported by the small data lab at Cornell Tech, which received funding from NSF, NIH, RWJF, UnitedHealth Group, Google, and Adobe.

TABLE OF CONTENTS

Biographical Sketch	iii
Dedication	iv
Acknowledgements	v
Table of Contents	vii
List of Tables	x
List of Figures	xii
1 Introduction	1
1.1 Debiasing implicit feedback	3
1.2 Leveraging richer data sources	3
1.3 Interactive preference learning	4
1.4 Generalization of recommendation algorithms	6
1.5 Thesis organization	7
2 Debiasing implicit feedback: unbiased recommender evaluation	8
2.1 Introduction	8
2.2 Related work	10
2.2.1 Debiasing the evaluation of ExplicitRec	11
2.2.2 ImplicitRec and evaluation	11
2.2.3 Counterfactual evaluation	12
2.3 Unbiased recommender evaluation for implicit feedback	12
2.3.1 Average-over-all (AOA) evaluator	14
2.3.2 Unbiased evaluator	16
2.3.3 Estimating propensity scores	17
2.4 Experiments with biased feedback and the evaluator	19
2.4.1 Experimental setup	20
2.4.2 Investigating popularity bias	24
2.4.3 Exploring the power-law exponent	26
2.4.4 Understanding the unbiased evaluator	28
2.5 Evaluating debiasing performance	30
2.5.1 Experimental setup	30
2.5.2 Results	31
2.6 Conclusions and discussions	32
3 Leveraging richer data sources: personalized creative applications incorporating unstructured software usage traces	33
3.1 Introduction	33
3.2 Related work	36
3.2.1 Distributed representation learning	36
3.2.2 User modeling in online social platforms	37
3.2.3 Software user and command modeling	37
3.3 Dataset	39

3.4	Software user representation	40
3.4.1	util2vec framework	40
3.4.2	Implementation details	42
3.4.3	User profiling performance	43
3.5	Applications	45
3.5.1	Software user tagging	47
3.5.2	Cold-start art project recommendation	52
3.5.3	Inspiration engine	58
3.6	Conclusions	61
4	Interactive preference learning: a personalized nutrient-based recipe recommendation system	62
4.1	Introduction	62
4.2	Related work	66
4.2.1	Healthy meal recommender system	66
4.2.2	Cold-start problem and preference elicitation	67
4.2.3	Pairwise algorithms for recommendation	68
4.2.4	Food image analysis	69
4.3	Yum-me system design	70
4.3.1	Large scale food database	71
4.3.2	User survey	74
4.3.3	Adaptive visual interface	75
4.4	Online learning framework	76
4.4.1	User state update	77
4.4.2	Images selection	81
4.5	FoodDist: food image embedding	83
4.5.1	Learning with classification	84
4.5.2	Metric learning	85
4.5.3	Multitask learning	87
4.6	Evaluation	88
4.6.1	User testing for the online learning framework	89
4.6.2	Offline benchmarking for FoodDist	94
4.6.3	End-to-end user testing	97
4.7	Discussions	106
4.7.1	Limitations of the evaluations	106
4.7.2	Limitations of Yum-me in recommending healthy meals	107
4.7.3	Yum-me for real world dietary applications	107
4.7.4	FoodDist for food image analysis tasks	108
4.8	Conclusions	109
5	Interactive preference learning: an intention-informed spoken word content recommendation system	110
5.1	Introduction	110
5.2	Related work	113

5.2.1	Effects of recommendations	113
5.2.2	User intentions	115
5.2.3	Recommendations beyond accuracy	116
5.2.4	Web spoken word content	116
5.3	Study design	117
5.3.1	Onboarding (ONB)	118
5.3.2	Field study (FIE)	120
5.3.3	Post-study survey	123
5.3.4	Participant recruitment	124
5.4	Study results	125
5.4.1	General usage patterns	126
5.5	Qualitative usage results	128
5.5.1	Choices related to topic-wise intentions	129
5.5.2	Exploratory choices	131
5.5.3	User satisfaction	133
5.6	Implications and discussions	134
5.6.1	Employing planning and intentions	134
5.6.2	Encouraging exploration	135
5.6.3	Understanding user satisfaction	136
5.6.4	Optimizing for multiple objectives	137
5.6.5	Limitations of intention-agnostic metrics	137
5.7	Conclusions	138
6	Generalization of recommendation algorithms	139
6.1	Introduction	139
6.2	Evolution of recommender systems	143
6.2.1	Pure collaborative filtering models	143
6.2.2	Hybrid and content-ware models	144
6.3	Related frameworks	145
6.4	OpenRec framework	147
6.4.1	Recommenders	148
6.4.2	Modules	150
6.4.3	Utility functions	152
6.4.4	Generalization	153
6.5	Experiments and use cases	153
6.5.1	Validity: reproducing monolithic implementations	154
6.5.2	Efficiency: quick prototyping and experimentation	160
6.5.3	Extensibility: developing new algorithms via extension	165
6.6	Conclusions	167
7	Future work	168
	Bibliography	171

LIST OF TABLES

2.1	The true and estimated DCG values for three recommenders in Fig. 2.1. $R(\hat{Z})$ denotes the ground truth. $\hat{R}_{\text{AOA}}(\hat{Z})$ denotes the AOA estimations. The AOA estimator outputs larger values when popular items are ranked higher.	15
2.2	Estimated γ value for every dataset–algorithm pair. The algorithm that achieves the lowest γ in each dataset is shown in boldface. The γ estimation is more sensitive to the choice of datasets than to the choice of algorithms.	27
2.3	The Mean absolute error (MAE) between evaluators’ outputs on the biased-testing set and recommenders’ true performances. Performance was measured against AUC and Recall. For the unbiased evaluator (UB), four γ values were used in the experiments ($\gamma = 1.5, 2.0, 2.5, 3.0$).	31
3.1	User fingerprinting performance (hold-out session retrieval) in terms of mean reciprocal rank (MRR). The improvement is relative to the <i>bag-of-actions+tf-idf</i>	45
3.2	5-nearest neighbors of selected actions according to the action embedding X . The actions in the first row are the index, and the five actions below are the corresponding nearest neighbors. (From left to right, the actions are related to <i>video editing</i> , <i>font awesome icons</i> , <i>blur filters</i> , <i>path manipulations</i> and <i>shadow effects</i> , respectively.)	46
3.3	User tagging performance in terms of Recall@K. We use boldface for the best performed approach and feature set. The percentage of improvements are the comparison between util2vec (boldface) and popular tags. Our tagging system outperforms the popularity tags baseline by 31.0% and 35.0% in terms of Recall@1 and Recall@2 respectively.	50
3.4	Art project recommendation performance for cold-start users in terms of Recall@K and AUC. We use boldface for the best performed approach and feature set. The percentage of improvements are the comparison between the approach in boldface and the baseline method (popular items).	57
3.5	Action-image retrieval performance in terms of Recall@K. We use boldface for the best performed approach.	59
4.1	The size of databases for different diet types. Unit: number of unique recipes.	72
4.2	Average duration to complete the training phase.	93
4.3	The classification task performance. * represents the state-of-the-art approaches, and the boldface text indicates the method with the best performance.	96

4.4	The retrieval task performance. * represents the state-of-the-art approaches, and the boldface text indicates the method with the best performance. (Note: The mAP value that we report for Food-CNN is higher because we use pixel-wise mean subtraction, whereas the original paper only used per-channel mean subtraction.)	97
4.5	The statistics of nutritional expectations indicated by 60 participants. Unit: number of participants.	100
4.6	The Average Acceptance Rates (Avg. Acc.), Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) of two recommendation systems. Paired t-test p-value (Avg. Acc.): $< 10^{-9}$	102
5.1	Participants' demographic information including gender, age, primary mobile device, and college major.	124
6.1	Comparing OpenRec to existing software frameworks for recommender systems (Sys-m: system-level modularity, Algo-m: algorithm-level modularity).	146
6.2	Testing performance on citeulike dataset [162] in terms of AUC and Recall@K (CDL-OpenRec and CDL-Original).	159

LIST OF FIGURES

2.1	A hypothetical example to illustrate the evaluation bias that results from use of the AOA evaluator. Three recommenders generated distinct lists of recommendations, Z^1 , Z^2 and Z^3 , for the same user. Among the shaded items that were preferred by the user, the ones with a solid border were observed by recommenders. The performance was measured by DCG, and the results are presented in Table 2.1.	14
2.2	The distribution of n_i^* (the observed number of interactions with item i) in the three datasets. The items are presented in descending order of n_i^* . The horizontal axis is log scaled for better visualization. In all datasets, the n_i^* distribution is skewed and the user interactions are significantly biased.	24
2.3	Empirically estimated $f(n^*)$ on the three datasets and the four recommendation algorithms. $f(n^*)$ denotes the average number of times that an item with observed popularity n^* was recommended. Both axes are log scaled. Therefore, exponential growth is linear in the figure. All settings manifest significant presentation bias.	25
2.4	Comparison of the traditional and unbiased evaluators in measuring the performance of four recommendation algorithms. The evaluations were conducted over three datasets using four metrics. Each sub-figure represents a specific dataset-metric pair. For the unbiased evaluator, three estimated γ values from Section 2.4.3 were used in the experiments. The unbiased evaluator significantly reduces the biased weights that the AOA method places on the popular items and produces robust and consistent results for any γ from the estimated range.	28
3.1	Services that can benefit from the integration of application usage traces. By leveraging the user representations learned from sequences of actions, software service providers and social platforms can improve their personalized services and empower new user experiences.	34
3.2	Data samples of Photoshop usage records (left) and social interactions on Behance (right). We collected three categories of social interactions for each user: <i>projects viewed</i> , <i>self-disclosed areas of focus</i> and <i>uploaded projects</i>	40
3.3	The architecture of util2vec model. The columns of V and X store the user representations and action representations respectively. While the action embedding X is shared across different users, user embedding V is user-specific.	41

3.4	Six user tagging examples with two different approaches. For each user, we show her portfolio, top 5 tag predictions with util2vec feature, top 5 most popular tags and self-disclosed tags. The tags with orange color are the correct predictions, and the ones with green color are the ones that are inferrable from the portfolio but not explicitly selected by the user.	51
3.5	Two-phase recommendation framework. In step 1, we derive users' latent factors and items' latent factors and bias from their implicit feedback (project views). In step 2, we learn a projection function f to map software usage features to the corresponding users' latent factors.	54
3.6	The algorithm framework for the inspiration engine. We learn a function g to project image features to the util2vec embedding space such that true actions-image pairs are close to each other and false pairs are farther away.	58
3.7	Four image retrieval results of the inspiration engine using single action. The retrieval results reflect the context where each action is often used. For example, with <i>fade_smart_blur</i> , returned images have blurred background and fading effects, and with <i>rotate_canvas</i> , images tend to have repetitive patterns.	60
4.1	An overview of Yum-me. This figure shows three sample scenarios in which Yum-me can be used: desktop browser, mobile, and smart watch. The fine-grained dietary profile is used to re-rank and personalize recipe recommendations.	70
4.2	An overview of two sample databases: (a) for users without dietary restrictions and (b) for vegetarian users.	73
4.3	User-system interaction at iteration t	76
4.4	The locally connected graph with item i	79
4.5	An Euclidean distance embedding of FoodDist. This figure shows the pairwise euclidean distances between food image representations. A distance of 0.0 means that two food items are identical and a distance of 2.0 represents that the image contents are completely different. In this example, if the threshold is set to 1.0, then all the images can be correctly classified.	83
4.6	The multitask learning structure of FoodDist. Different types of layers are denoted by different colors. The format of each type of layer: Convolution layer: [<u>receptive field size:step size...</u> , <u>#channels</u>]; Pooling layer: [<u>pooling size:step size...</u>]; Fully connected layer: [<u>..., output dimension</u>].	86
4.7	The prediction accuracy of different algorithms in various training settings (asterisks represent different levels of statistical significance: *** : $p < 0.001$, ** : $p < 0.01$, * : $p < 0.05$).	91

4.8	The cumulative distribution of the prediction accuracy of <i>LE+EE</i> algorithm (Numbers in the legend represent the values of <i>T</i>). . .	91
4.9	Comparison of the cumulative distribution of prediction accuracy across different algorithms.	92
4.10	User response time and system execution time.	93
4.11	The survey used for user onboarding at PlateJoy. (The top four questions are included.)	98
4.12	The workflow of the end-to-end user testing. We compare Yum-me (blue arrows) to the baseline method (violet arrow) that makes recommendations solely based on nutritional expectations and dietary restrictions.	99
4.13	The cumulative distribution of the acceptance rate for both recommender systems.	101
4.14	The distribution of the acceptance rate difference between two recommender systems. The difference is normally distributed (A Shapiro Wilk W test is not significant ($p = 0.12$)), and a paired Student's t-test indicates a significant difference between the two methods ($p < 0.0001$).	102
4.15	Comparison of nutritional facts among participants' favorite recipes, accepted Yum-me recommendations, and accepted baseline recommendations. The recipe is accepted if it was dragged into the <i>yummy</i> bucket. The mean values are normalized by the average amount of corresponding nutrient in the favorite recipes (orange bar). (Only 7 out of 9 nutritional goals were chosen by at least one participant.)	103
4.16	A qualitative analysis of Yum-me recommendations. Images on the left half are sampled from users' top-20 favorite recipes; Images on the right half are the ones recommended to the users. The number under each food image corresponds to the amount of calories, unit: <i>kcal/serving</i>	104
4.17	The entropy of the preference distributions in different iterations of online learning. (Using data from 48 participants with no dietary restrictions.)	105
5.1	The web user interface designed for participants to indicate their topic-wise intentions. Participants first select up to eight general topics they want to listen to and then optionally select fine-grained topics. The topics are defined using podcast categories in iTunes.	118

5.2	The web user interface designed for participants to subscribe to channels during onboarding. The interface presented a list of podcast shows, and participants were instructed to subscribe to up to ten of them. For the control group (POP), channels were ordered by their popularity on iTunes, whereas for the experimental group (ASP), the ordering was determined by channels' alignment to participants' topic-wise intentions. Both groups shared the same set of candidate content.	119
5.3	The library and home pages of the customized podcast mobile app. The library page showed the channels to which a user has subscribed, and the home page chronologically presented a list of episodes. For the control group (SUB), the episodes were retrieved from subscribed channels, whereas for the experimental group (MIX), those episodes were mixed with the ones selected from not-subscribed channels based on a CF model.	121
5.4	The discover page of the customized podcast mobile app. The page grouped channels into topic-wise categories and presented a trending chart that ordered channels according to their popularity on iTunes. This page allowed users to readily explore and subscribe to new channels.	121
5.5	The cumulative distributions of users over the number of subscriptions and listening time. These figures show the extent to which participants were actively subscribing and listening to podcasts throughout the study. A vertical line in these figures represents a group of users with a similar activity level. We note that these commonly-used aggregated measures are not statistically different across the four groups. In other words, they do not reflect the different composition of content consumption across these groups. These differences are critical to understand the effects of recommendations on individual growth and experience.	125
5.6	The distribution of podcast listening instances over hour of day and day of week. The aggregation is across all participants. Again we note that no statistical difference is observed across the four groups.	127
5.7	The cumulative distributions of subscriptions and listening time over channels ordered by popularity. The popularity is defined as the number of subscription (a, b) and the amount of listening (c). These figures show the extent to which participants' content consumption was concentrated on a small set of popular items. A linear line in the figure represents uniformly distributed consumption over all channels. During onboarding, the POP intervention resulted in significant popularity bias in participants' subscriptions, but in the field, no significant effect from experimental factors is observed.	128

5.8	The top five most interacted content source during onboarding and in the field, categorized by 2×2 groups. Each square icon represents a podcast channel. These qualitative results demonstrate how users' content consumption in the field was jointly affected by users' intentions and recommendation systems. . . .	129
5.9	The distribution of user intentions over podcast topics (categories). Topics are sorted by their popularity descendingly. Participants' intended topics were diversely spread, with most of the topics liked by less than half of the participants.	129
5.10	The cumulative distributions of users over the percentage of the topicwise intention-related subscriptions and listening. In the above figures, an $x = 1.0$ curve denotes that all users' consumption is related to their topicwise intentions, while an $x = 0.0$ curve denotes that none is related. The ASP intervention during onboarding is shown to significantly increase the topic-related onboarding subscriptions, topic-related field subscriptions, and topic-related field listening. The FIE factor and the interaction $ONB \times FIE$ have no significant effect.	130
5.11	The groupwise average percentage of the topicwise intention-related subscriptions and listening. The ASP intervention significantly improves the topic-relatedness of onboarding subscriptions, field subscriptions, and field episode listening by 72.1%, 36.5%, and 24.9% respectively. The FIE and the interaction ($ONB \times FIE$) have no significant effect.	131
5.12	The percentage of subscriptions and listening from not-subscribed channels: (a) cumulative distributions over users, and (b) groupwise average. In (a), a $x = 1.0$ curve denotes that users do not listen to episodes from subscribed channels, while a $x = 0.0$ curve denotes that all listening comes from subscribed channels. The MIX intervention is shown to significantly increase the exploration rate by 127.5%. The ONB and the interaction ($ONB \times FIE$) have no significant effect.	132
5.13	Participants' satisfaction (the averaged ratings of all indicators): (a) cumulative distributions of aggregated ratings, and (b) groupwise average ratings. The interaction between two factors ($ONB \times FIE$) significantly affects satisfaction — MIX improves satisfaction if participants were onboarded with the ASP, otherwise MIX shows negative effects. No single factor alone has a significant effect.	133

6.1	A modular view of recommendation algorithms. Each algorithm (R-1 to R-4) is a structured ensemble of reusable modules under three categories: extraction module , fusion module , and interaction module . The color codex is shared throughout this chapter. Arrows in the figure represent data flows.	142
6.2	The architecture of OpenRec. A recommender is built out of modules. All three components (Module, Recommender, and Utility) can be seamlessly used together to conduct training, evaluation, experimentation, and serving of recommenders. . . .	147
6.3	Standard interfaces of the Recommender abstraction. It contains procedures for constructing computational graphs, and functions for model training, testing, saving and loading.	148
6.4	The structure of the Module abstraction (Left: inputs, Right: outputs).	149
6.5	Implementing BPR with OpenRec (45 lines). We use rectangles to represent functions in a Recommender and shade the reusable modules and implementations. An arrow denotes an adoption or an inheritance. (Lines of code does not include blank and import lines.)	155
6.6	Testing performance on <i>tradesy.com</i> dataset [67] in terms of AUC (BPR-OpenRec, BPR-original, and BPR-MyMediaLite).	156
6.7	Implementing VBPR with OpenRec (50 lines). We use the same annotations as Fig. 6.5.	157
6.8	Testing performance on <i>tradesy.com</i> dataset [67] in terms of AUC (VBPR-OpenRec and VBPR-original).	158
6.9	Implementing UserVisualPMF with OpenRec (32 lines). We use the same annotations as Fig. 6.5.	161
6.10	Testing performance in book recommendations in terms of AUC and Recall@K.	163
6.11	Implementing VisualCML with OpenRec (7 lines). We use the same annotations as Fig. 6.5. The model and training pseudocode are presented in Listing 1 and 2 respectively.	163
6.12	Implementing an iterative and temporal model with OpenRec (57 lines). We use the same annotations as Fig. 6.5.	165

CHAPTER 1

INTRODUCTION

Recommendation systems (recommenders) are algorithmic modules that select and present information items to individuals in a personalized fashion. These modules have been widely incorporated into a variety of digital services, including e-commerce websites, content platforms, and education and wellness applications. Beyond their significant commercial value, recommenders have also implicitly modulated people’s daily actions and choices. For example, the merchandise shown by e-commerce platforms (e.g., Amazon, Etsy, and FreshDirect) can influence people’s buying and eating decisions, the information presented by content services (e.g., YouTube, Facebook, and Twitter) may affect the activities that people choose to engage in, and the suggestions made by online education and career websites (e.g., Coursera and LinkedIn) tend to modulate the courses that students take and the jobs to which they apply. This potential impact on individuals challenges the design of recommendation systems to balance commercial interests and individuals’ utility. This thesis addresses the research question of how to build **user-centric recommendation systems** optimized for end-user benefit, which complements the traditional objective of promoting commercial metrics, such as sales and click-through rates.

Traditional commercial recommenders are optimized to promote users’ implicit feedback, which refers to users’ actions that reveal their positive preferences, such as click [73], view [174], watch [32], listen [158], share [63], etc. These systems take a user’s feedback history as input and predict how likely the individual is to interact with each item positively. The higher-scoring items are then recommended first. While implicit feedback signals are advantageous for

their large volume and wide availability, they are limited in capturing recommenders' utility to individuals for two main reasons. First, **implicit feedback is biased**. Users can only interact with the items that they see [131, 172]. In reality, the items presented to users are not randomly chosen but rather selected by systems optimized for biased metrics, such as inferred user preferences, key performance indicators (KPIs), etc. Therefore, implicit feedback is not missing at random. Second, **implicit feedback provides an incomplete view of user preferences**. On the one hand, cold-start users can only be partially profiled using implicit feedback [72, 130, 188] because newcomers often have limited or no interactions with a platform. On the other hand, implicit feedback does not reflect whether or not recommendations are aligned with users' aspirational preferences [179], even though they may engage users at present.

In light of the above limitations, this thesis explores three research directions to empower user-centric recommendation systems: (1) debiasing implicit feedback, (2) leveraging richer data sources, and (3) interactive preference learning. In addition, this thesis designs and introduces an open-source tool that generalizes application-specific recommendation algorithms through modularization.

Beyond commercial interests and individuals' utility, recommendation systems also play important roles in societal issues, such as political polarization, fairness, and social welfare. These topics are critically important directions for future research, but they are beyond the scope of this thesis. We hope this thesis can open up future explorations of recommenders that contribute to both societal and individual good.

The following are further summaries of our contributions.

1.1 Debiasing implicit feedback

Implicit feedback is Missing-Not-Completely-At-Random (MNCAR) because online platforms are subject to popularity bias (i.e., popular items are more likely to be presented and interacted with). As a result, when it comes to evaluating recommendation algorithms, the widely used Average-Over-All (AOA) approach is biased toward accurately recommending popular items [172]. In this thesis, we investigate the bias of AOA and develop a debiased and practical offline evaluator for implicit MNCAR datasets using the Self-Normalized Inverse-Propensity-Scoring (SNIPS) [147] technique. Through extensive experiments using four real-world datasets and four widely used algorithms, we show that (1) popularity bias widely manifests in item presentation and interaction; (2) evaluation bias due to MNCAR data pervasively exists in most cases where AOA is used to evaluate implicit-feedback based recommenders; and (3) the debiased estimator significantly reduces the bias of AOA by more than 30% in the Yahoo music dataset in terms of Mean Absolute Error (MAE).

1.2 Leveraging richer data sources

Implicit feedback is merely the tip of the iceberg in the sea of digital traces that individuals generate. From topics referred to in social media or email, to photos captured through smartphones, and to software usage history, these rich data sources are potentially less biased and more comprehensive in reflecting who we are and what we are interested in [72, 48, 174]. In this thesis, we ex-

plore the utility of unstructured log-trace data generated by users of software applications [174, 175]. These traces contain hidden clues to the intentions and interests of those users, but service providers may find it challenging to uncover and exploit them. We propose a framework for personalizing software and web recommendation systems by leveraging such unstructured traces. We use six months of Photoshop usage history and seven years of interaction records from 67,000 Behance users to design, develop, and validate a user-modeling technique (which we call the **utilization-to-vector** or **util2vec** model) that discovers highly discriminative representations of Photoshop users. We demonstrate the promise of this approach on three exemplary recommenders: (1) a practical user tagging system that predicts areas of focus for millions of Photoshop users; (2) a two-phase recommendation model that enables cold-start personalized recommendations for new Behance users who have Photoshop usage data, which improves recommendation quality (Recall@100) by 21.2% over a popularity-based recommender; and (3) a novel inspiration engine that provides real-time personalized inspirations to artists.

1.3 Interactive preference learning

Passively collected offline data is limited when it comes to modeling cold-start users and serving users’ aspirational preferences. For example, in the domain of diet and nutrition, food-journaling-based meal recommenders often require a prolonged learning period because of the significant journaling burden, and they are agnostic about people’s health objectives; similarly, on content platforms, the information that users consumed in the past may not align with their aspirations. In this thesis, we address these problems by developing interactive

systems to elicit users’ current and aspirational preferences that are not reflected in the offline data. Specifically, we design, build and evaluate two systems.

The first system, named **Yum-me** [173, 177], is a personalized nutrient-based recipe recommender system designed to meet individuals’ nutritional expectations, dietary restrictions, and fine-grained food preferences. Yum-me enables a simple and accurate food preference profiling procedure via a visual quiz-based user interface. The system conducts recommendations by leveraging the learned profile to re-rank nutritionally appropriate food options. Our design of Yum-me makes two main research contributions: (1) an open source state-of-the-art food image analysis model, named **FoodDist**, that can be used for a wide variety of dietary applications; and (2) a novel online learning framework that learns food preferences from item-wise and pair-wise image comparisons. In an online study with 227 anonymous users, the framework outperformed other baselines by a significant margin. We conducted an end-to-end evaluation of Yum-me through a 60-person lab study, where Yum-me improved the recommendation acceptance rate by 42.63% without sacrificing nutrition-wise performance, as compared to existing systems that only consider people’s nutritional needs.

The second system is an intention-informed recommendation system [179] for spoken word content (podcasts) [180]. We modify a commercial podcast app to include a recommender that elicits users’ stated intentions at onboarding, and a collaborative filtering (CF) recommender during daily use. To compare the effects of intention informed recommenders with classical intention agnostic systems, we conducted a 2×2 randomized controlled field experiment with 105 participants. Our study suggests that: (1) Intention-aware recommenda-

tions can significantly raise users’ interactions (subscriptions and listening) with channels and episodes related to intended topics by over 24%, even if such a recommender is only used during onboarding. (2) The CF-based recommender doubles users’ explorations on episodes from not-subscribed channels and improves satisfaction for users onboarded with the intention-aware recommender.

1.4 Generalization of recommendation algorithms

State-of-the-art recommendation systems, including user-centric recommenders, have gone beyond simple user-item filtering and are increasingly sophisticated, comprised of multiple components for analyzing and fusing diverse information. Unfortunately, existing frameworks do not adequately support extensibility and adaptability; consequently, they pose significant challenges to rapid, iterative, and systematic experimentation. In this thesis, we design **OpenRec** [171], an open and modular Python framework that supports extensible and adaptable research in recommender systems. Each recommender is modeled as a computational graph that consists of a structured ensemble of reusable modules connected through a set of well-defined interfaces. We demonstrate that OpenRec provides adaptability, modularity, and reusability while maintaining training efficiency and recommendation accuracy. Our case study illustrates how OpenRec can support an efficient design process to prototype and benchmark alternative approaches with interchangeable modules and enable the development and evaluation of new algorithms.

1.5 Thesis organization

This thesis is organized into seven chapters. Chapter 2 presents our algorithm to infer debiased user preferences from biased implicit feedback data. Chapter 3 develops techniques to learn broader user preferences from richer data sources for creative professionals. Chapter 4 and Chapter 5 introduce interactive systems that elicit user preferences not reflected in passively collected offline data. Chapter 4 focuses on recommending favorable and nutritionally appropriate recipes, whereas Chapter 5 aims at incorporating users' aspirations into spoken word content recommendations. Lastly, Chapter 7 concludes the thesis with discussions of future research plans.

CHAPTER 2

DEBIASING IMPLICIT FEEDBACK: UNBIASED RECOMMENDER EVALUATION

2.1 Introduction

This chapter addresses the research question of how to eliminate the biases in implicit feedback to improve the evaluations of recommendation algorithms. Unlike other machine learning applications, recommenders are notoriously challenging to evaluate offline because of the biases in user feedback data. Prior work on Explicit-rating Recommenders (**ExplicitRec**) [185, 103] revealed that users give subjective ratings to items, which results in Missing-Not-Completely-At-Random (**MNCAR**) ground truth data. It has been widely recognized in the literature [131, 140, 139, 138, 103] that MNCAR rating data can lead to biased conclusions. Therefore, many mechanisms are proposed to debias offline recommender evaluation of rating data [131, 140, 139, 138].

However, existing approaches are not directly applicable to implicit feedback, which are much more prevalent and have been widely used by many state-of-the-art recommendation solutions [71, 36, 174]. Different from explicit ratings (e.g., those based on a Likert scale), implicit feedback signals are one-sided and positive only. In other words, an ideal recommender would never observe user interactions with *irrelevant*¹ items, whereas in ExplicitRec, complete observations assume that each user has a latent preference score for every item. As a result, for Implicit-feedback Recommenders (**ImplicitRec**), it is un-

¹An item is *relevant* to a user if the user is interested in interacting with it (e.g., clicking or viewing it). Otherwise, the item is regarded as *irrelevant*.

clear whether a missing item in a user’s history is not favored by the user or has simply not yet been observed.

Existing work simplifies the evaluation of ImplicitRec by assuming that positive signals are Missing-Completely-At-Random (**MCAR**) [98, 67, 71], i.e., each favored item is equal-likely to be clicked or viewed by a user. This assumption does not hold in real-world settings because online recommenders manifest popularity bias [2] (popular items are much more likely to be recommended and presented to users). Such a bias leads to the phenomenon that trendy items are more likely to be interacted with by users. Eventually, the Average-Over-All (**AOA**) evaluator implicitly places greater weights on the accuracy of serving popular items than on serving long-tail ones. This may overlook key limitations of recommendation algorithms, such as under-serving cold start groups [158], being dominated [2], and exacerbating unhealthy user behavior [154]

To address the MNCAR nature of implicit feedback, we develop an algorithmic framework based on the Inverse-Propensity-Scoring (**IPS**) technique used in causal inference [80], which was recently applied to evaluate ExplicitRec [131]. Specifically, we (1) qualitatively and theoretically demonstrate that the existing evaluation protocol for ImplicitRec is biased; (2) derive unbiased performance estimators for major evaluation metrics, including AUC, DCG, DCG@K, and Recall@K; and (3) conduct extensive experiments using four real-world datasets (citeulike [162], Tradesy [67], Amazon book [105, 171], and Yahoo music [169]) and four widely used algorithms (BPR [124], PMF [129], U-CML [71], and A-CML [71]). Our experimental results highlight three key contributions and implications of this chapter:

- The analysis of datasets and trained models (Section 2.4.2) reveals that

popularity bias is widely manifested in item presentation (i.e., popular items are more likely to be presented than long-tail ones) and interaction (i.e., users tend to interact more with popular items). This implies that more attention is needed in considering the potentially negative social and economic impacts of the bias [2, 154].

- The comparisons of the classical AOA evaluator to the unbiased evaluator proposed herein (Sections 2.4.3 and 2.4.4) demonstrate that AOA is biased in evaluating most ImplicitRec. The bias may lead to inaccurate judgments of algorithmic improvements and sub-optimal decisions when it comes to model selection.
- The unbiased evaluator significantly reduces AOA evaluation error by more than 30% in the Yahoo music dataset in terms of the mean absolute error (MAE) (Section 2.5).

Our code is available at <https://github.com/ylongqi/unbiased-offline-recommender-evaluation>.

2.2 Related work

This chapter is inspired by three lines of research: (1) debiasing the evaluation of ExplicitRec, (2) ImplicitRec algorithms and evaluations, and (3) counterfactual evaluation. In this section, we discuss how this chapter builds upon existing ideas and contributes new knowledge to the field.

2.2.1 Debiasing the evaluation of ExplicitRec

Previous research has shown that for explicit-feedback recommenders, users' ratings are MNCAR [131, 140, 139, 138, 103]. This is because people tend to subjectively choose the items they rate, and the selection reflects biases of personal preferences [131] and opinions [103, 140]. To handle MNCAR data and conduct unbiased evaluation, previous work assumed that users have latent ratings for every item, and then use popularity [139] or other predictive models [131] to estimate the probability that any given rating is observed. However, such a paradigm is not applicable to implicit feedback, because of two fundamental differences: Implicit feedback (1) is available only for the subset of items preferred by users, and (2) is often recorded passively and thus is unlikely to be intentionally controlled.

This chapter addresses the unique missing patterns of implicit feedback by extending the IPS framework [131].

2.2.2 ImplicitRec and evaluation

Recently, there has been a trend toward development of recommenders using implicit feedback signals [73], such as click [162, 71], watch [36], and view [174]. These signals are much richer than ratings. Classical offline evaluation approaches [98, 67, 162, 71, 174] randomly hold out one interacted item per user as a testing set and then report the average performance. Such a paradigm has been shown to be unbiased under MCAR feedback [98]. However, MCAR signals rarely exist in the real world, because it is very unlikely that a content platform would present items completely at random. In fact, item presentation is

often mediated by recommenders that are subject to popularity bias [2].

This chapter points out that under MNCAR user feedback, the existing evaluation paradigm is biased. In light of this, we develop a practical and effective technique to address the bias.

2.2.3 Counterfactual evaluation

Our unbiased evaluator is based on the techniques developed for counterfactual evaluation [146, 148, 80], which aim to evaluate ranking policies offline based on the logs collected from online interactive systems. It has been successfully applied to interactive search [81] and recommendation [96, 148]. Our debiasing framework is built on the Self-Normalized Inverse-Propensity-Scoring (**SNIPS**) estimator proposed by Swaminathan et al. [147].

However, classical counterfactual reasoning operates on interactive logs, for example, $(\text{user}_1, \text{article}_1, \text{reward}_1), \dots, (\text{user}_n, \text{article}_n, \text{reward}_n)$, which are different from the implicit feedback-based matrix completion task that we consider. To the best of our knowledge, there has been little research on applying counterfactual estimators to debias ImplicitRec evaluations.

2.3 Unbiased recommender evaluation for implicit feedback

Recommenders built on implicit feedback receive only users' one-sided (positive) preference signals, such as clicks and watches. Under complete observations, user u has a set of preferred items S_u among the entire set of items, \mathcal{I} (i.e.,

$\mathcal{S}_u \subseteq \mathcal{I}$). An ideal recommendation evaluator calculates the following reward $R(\hat{Z})$ for the predicted item ranking \hat{Z} .

$$R(\hat{Z}) = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|\mathcal{S}_u|} \sum_{i \in \mathcal{S}_u} c(\hat{Z}_{u,i}), \quad (2.1)$$

where $\hat{Z}_{u,i}$ is the predicted ranking of item i (among all the items in \mathcal{I}) for user u , and the function c denotes any top-N scoring metric, such as area under the ROC curve (AUC), discounted cumulative gain (DCG), DCG@K, or Recall@K. These functions are defined as follows:

$$\text{AUC: } c(\hat{Z}_{u,i}) = 1 - \frac{\hat{Z}_{u,i}}{|\mathcal{I}|} \quad (2.2)$$

$$\text{DCG: } c(\hat{Z}_{u,i}) = \frac{1}{\log_2(\hat{Z}_{u,i} + 1)} \quad (2.3)$$

$$\text{DCG@K: } c(\hat{Z}_{u,i}) = \frac{\mathbf{1}\{\hat{Z}_{u,i} \leq K\}}{\log_2(\hat{Z}_{u,i} + 1)} \quad (2.4)$$

$$\text{Recall@K: } c(\hat{Z}_{u,i}) = \mathbf{1}\{\hat{Z}_{u,i} \leq K\} \quad (2.5)$$

Eqn. 2.1 measures idealistic recommendation performance, which assumes that users would go through all items in the system and interact with every one that appeals to them. From a practical standpoint, it is impossible to browse and judge millions or billions of items. As a result, recommenders have access to only a partial view of \mathcal{S}_u , denoted by \mathcal{S}_u^* . For each positive signal $(u, i), i \in \mathcal{S}_u$, we use $O_{u,i}$ to indicate whether (u, i) is observed ($O_{u,i} = 1$ if (u, i) is observed, and $O_{u,i} = 0$ otherwise). In addition, inspired by [131], we assume the observations of every signal to be Bernoulli distributed, i.e., $O_{u,i} \sim \mathcal{B}(1, P_{u,i})$, where with probability $P_{u,i} = P(O_{u,i} = 1)$, (u, i) is observed by a recommender.

In reality, the partial view \mathcal{S}_u^* is mostly biased and the implicit feedback is MNCAR. In Section 2.3.1, we show that the AOA evaluator, which is widely used in the existing literature, is biased, and in Section 2.3.2 we propose an unbiased evaluator based on the inverse-propensity-scoring (IPS) technique [131].

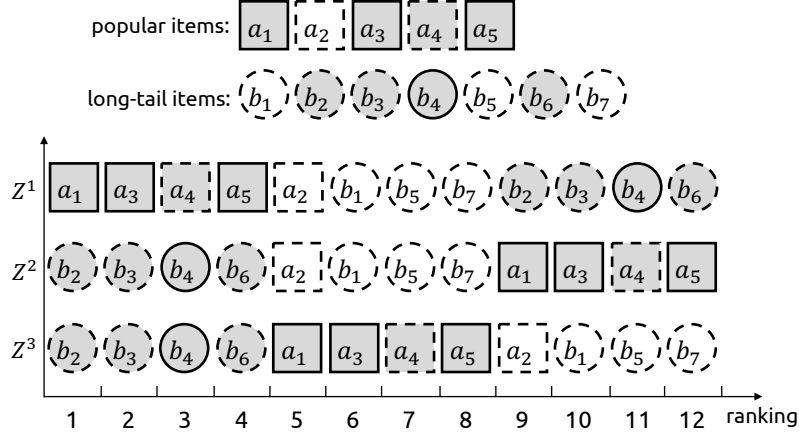


Figure 2.1: A hypothetical example to illustrate the evaluation bias that results from use of the AOA evaluator. Three recommenders generated distinct lists of recommendations, Z^1 , Z^2 and Z^3 , for the same user. Among the shaded items that were preferred by the user, the ones with a solid border were observed by recommenders. The performance was measured by DCG, and the results are presented in Table 2.1.

2.3.1 Average-over-all (AOA) evaluator

In prior literature, $R(\hat{Z})$ was estimated by taking the average over all observed user feedback S_u^* :

$$\begin{aligned} \hat{R}_{\text{AOA}}(\hat{Z}) &= \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|S_u^*|} \sum_{i \in S_u^*} c(\hat{Z}_{u,i}) \\ &= \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{\sum_{i \in S_u} O_{u,i}} \sum_{i \in S_u} c(\hat{Z}_{u,i}) \cdot O_{u,i} \end{aligned} \quad (2.6)$$

To intuitively illustrate the bias of the AOA evaluator, we considered a hypothetical platform that served 12 items, as shown in Fig. 2.1. We divided the items into two groups based on the number of interactions they received: popular items (a_1, \dots, a_5) and long-tail items (b_1, \dots, b_7). For a specific user, three different recommenders generated distinct ranked lists, Z^1, Z^2 , and Z^3 , based on the predicted user preferences. Each item on the platform was either relevant

Table 2.1: The true and estimated DCG values for three recommenders in Fig. 2.1. $R(\hat{Z})$ denotes the ground truth. $\hat{R}_{\text{AOA}}(\hat{Z})$ denotes the AOA estimations. The AOA estimator outputs larger values when popular items are ranked higher.

Estimator	Z^1	Z^2	Z^3
$R(\hat{Z})$	0.463	0.463	0.494
$\hat{R}_{\text{AOA}}(\hat{Z})$	0.585	0.340	0.390

(shaded) or irrelevant (blank) to the user. Among all the relevant items, only feedback for a partial set was observed (solid border). To encode the popularity bias manifested in ImplicitRec (i.e., user interactions with popular items are more likely to be observed), we assumed that among the relevant items, 75% of the popular items and 25% of the long-tail items were interacted with. In addition, three ranked lists were strategically designed: The Z^1 and Z^2 ranked lists had the same *true performance* on the ranking of *relevant* items but differed on the serving of the popular and long-tail groups. The Z^3 ranked list achieved the best *true performance*.

We calculated the DCG scores (eqn. 2.3) for three recommenders using the AOA evaluator (eqn. 2.6) and compared the scores to the true performances (eqn. 2.1). According to the results presented in Table 2.1, Z^1 was evaluated as much more accurate than Z^2 and Z^3 , despite the fact that, in reality, Z^2 had the same performance as Z^1 , and Z^3 performed much better. This demonstrates that the AOA evaluator is significantly biased toward the accuracy of serving trendy items; i.e., the estimated $\hat{R}_{\text{AOA}}(\hat{Z})$ is larger if popular items are ranked higher. The conclusions made based on such empirical evidence result in incorrect and even opposite judgments of the relative utilities of recommenders.

Basically, the expected outcome of the AOA evaluator does not conform to the true performance, i.e., $\mathbb{E}_O [\hat{R}_{\text{AOA}}(\hat{Z})] \neq R(\hat{Z})$. We prove this inequivalence by a counterexample. Suppose that for any user u , among all relevant items (\mathcal{S}_u), only one item $k^u \in \mathcal{S}_u$ has an observation probability close to 1, so that $P(O_{u,k^u}) = 1 - \epsilon$; whereas for the other items, $P(O_{u,i}) = \epsilon, i \in \mathcal{S}_u \setminus \{k^u\}$. In this case, $\mathbb{E}_O [\hat{R}_{\text{AOA}}(\hat{Z})] \approx_{\epsilon \ll 1} \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} c(\hat{Z}_{u,k^u}) \neq R(\hat{Z})$. Next, we present our proposed unbiased performance evaluator as an alternative to the existing AOA evaluator.

2.3.2 Unbiased evaluator

To conduct unbiased evaluation of biased observations, we leverage the IPS framework [131, 147] that weights each observation with the inverse of its propensity, where the term *propensity* refers to the tendency or the likelihood of an event happening. The intuition is to down-weight the commonly observed interactions, while up-weighting the rare ones. In the context of this chapter, the probability $P_{u,i}$ is treated as the pointwise propensity score. Therefore, the IPS unbiased evaluator is defined as follows:

$$\begin{aligned} \hat{R}_{\text{IPS}}(\hat{Z}|P) &= \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|\mathcal{S}_u|} \sum_{i \in \mathcal{S}_u} \frac{c(\hat{Z}_{u,i})}{P_{u,i}} \\ &= \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|\mathcal{S}_u|} \sum_{i \in \mathcal{S}_u} \frac{c(\hat{Z}_{u,i})}{P_{u,i}} \cdot O_{u,i} \end{aligned} \quad (2.7)$$

We prove that given any propensity assignment P , $\hat{R}_{\text{IPS}}(\hat{Z}|P)$ is unbiased.

$$\begin{aligned} \mathbb{E}_O [\hat{R}_{\text{IPS}}(\hat{Z}|P)] &= \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|\mathcal{S}_u|} \sum_{i \in \mathcal{S}_u} \frac{c(\hat{Z}_{u,i})}{P_{u,i}} \cdot \mathbb{E}_O [O_{u,i}] \\ &= \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|\mathcal{S}_u|} \sum_{i \in \mathcal{S}_u} c(\hat{Z}_{u,i}) = R(\hat{Z}) \end{aligned} \quad (2.8)$$

Furthermore, to estimate $|S_u|$ and control the variability of the IPS evaluator, we leverage the control variates [147, 131] to derive a Self-Normalized Inverse-Propensity-Scoring (SNIPS) evaluator. According to the theory of Monte Carlo approximation [147], the estimation \hat{W} of the expectation $\mathbb{E}_X[W(X)]$ has a lower variance if a multiplicative control variate $V(X)$ with known expectation $\mathbb{E}_X[V(X)] = v \neq 0$ is introduced, i.e., if \hat{W} is calculated as: $\hat{W} = \frac{\sum_{j=1}^n W(X_j)}{\sum_{j=1}^n V(X_j)} v$. While \hat{W} is not a completely unbiased estimator, it strongly converges to the true expectation for large n [147].

In the context of the IPS evaluator, because $\mathbb{E}_O\left[\sum_{i \in S_u^*} \frac{1}{P_{u,i}}\right] = \mathbb{E}_O\left[\sum_{i \in S_u} \frac{1}{P_{u,i}} \cdot O_{u,i}\right] = |S_u|$, we can write the SNIPS evaluation as follows:

$$\begin{aligned} \hat{R}_{\text{SNIPS}}(\hat{Z}|P) &= \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|S_u|} \frac{\mathbb{E}_O\left[\sum_{i \in S_u^*} \frac{1}{P_{u,i}}\right]}{\sum_{i \in S_u^*} \frac{1}{P_{u,i}}} \sum_{i \in S_u^*} \frac{c(\hat{Z}_{u,i})}{P_{u,i}} \\ &= \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{\sum_{i \in S_u^*} \frac{1}{P_{u,i}}} \sum_{i \in S_u^*} \frac{c(\hat{Z}_{u,i})}{P_{u,i}} \end{aligned} \quad (2.9)$$

A key challenge in computing $\hat{R}_{\text{SNIPS}}(\hat{Z}|P)$ is to predict the propensity scores $P_{u,i}$. Next, we demonstrate our method, which estimates the propensity scores based solely on raw observations, without requiring any auxiliary information.

2.3.3 Estimating propensity scores

We assume that the propensity score $P_{u,i}$ is user independent, i.e., $P_{u,i} = P(O_{u,i} = 1) = P(O_{*,i} = 1) = P_{*,i}$. This simplified assumption is made to address the lack of auxiliary user information in many user–item interaction records.² We derive $P_{*,i}$ by constructing a **two-step generative process** of user–item interactions: (1)

²This assumption may be relaxed in cases where auxiliary user information is available. We discuss this issue in Section 2.6.

Select, where a recommender system selects a set of items to present to a user; and (2) **Interact**, where the user browses the recommended items and interacts with the ones she likes. Therefore, $P_{*,i}$ can be calculated as follows:

$$P_{*,i} = P_{*,i}^{\text{select}} \cdot P_{*,i}^{\text{interact|select}}, \quad (2.10)$$

where $P_{*,i}^{\text{select}}$ is the probability that item i is recommended and $P_{*,i}^{\text{interact|select}}$ is the conditional probability that the user interacts with item i given that it is recommended.

Since implicit feedback is passively recorded and is less likely to be subjectively manipulated, we assume that $P_{*,i}^{\text{interact|select}} = P_{*,i}^{\text{interact}}$, i.e., the user interacts with all the items she likes in the recommended set, and the user's preferences are not affected by recommendations.³ Also, because $P_{*,i}^{\text{interact}}$ is user independent, it is proportional to only the item's *true popularity* n_i (the number of occurrences in the *complete observation*):

$$\hat{P}_{*,i}^{\text{interact}} \propto n_i \quad (2.11)$$

Because items that are frequently interacted with are more likely to be recommended in ImplicitRec [2], the probability $P_{*,i}^{\text{select}}$ is modeled using n_i^* (the number of times item i is interacted with) as a covariate. Specifically, we follow a common template that accurately captures the popularity bias [139], which assumes that $P_{*,i}^{\text{select}}$ conforms to a power-law distribution parameterized by γ :

$$\hat{P}_{*,i}^{\text{select}} \propto (n_i^*)^\gamma \quad (2.12)$$

Therefore, according to the constructed generation process, $\hat{P}_{*,i}$ depends on

³In reality, user-item interactions may be affected by the order of presentation of the items, and users' preferences may be shaped by recommendations in the long term. Modeling these effects may further improve the evaluator's performance (as discussed in Section 2.6).

only two variates, n_i^* and n_i :

$$\hat{P}_{*,i} \propto (n_i^*)^\gamma \cdot n_i, \quad (2.13)$$

where $n_i = \sum_{u \in \mathcal{U}} \mathbf{1}[i \in \mathcal{S}_u]$ and $n_i^* = \sum_{u \in \mathcal{U}, i \in \mathcal{S}_u^*} O_{*,i}$.

However, n_i is not directly observable. To address this problem, we observe that n_i^* is sampled from a binomial distribution⁴ parameterized by n_i , i.e., $n_i^* \sim \mathcal{B}(n_i, P_{*,i})$. Therefore, a relationship between n_i and n_i^* can be built by bridging the generative model (eqn. 2.13) with the following unbiased estimator:

$$\hat{P}_{*,i} = \frac{n_i^*}{n_i} \propto (n_i^*)^\gamma \cdot n_i \quad (2.14)$$

Therefore, $n_i \propto (n_i^*)^{\frac{1-\gamma}{2}}$. We use this as a replacement for the unobserved n_i in eqn. 2.13, which results in an unbiased $\hat{P}_{*,i}$ estimator that is determined by only the empirical counts of items:

$$\hat{P}_{*,i} \propto (n_i^*)^{\left(\frac{\gamma+1}{2}\right)} \quad (2.15)$$

Different values of the power-law exponent γ affect the propensity distributions over items with different observed popularity levels. A larger γ leads to lower propensity scores for long-tail items and higher scores for popular ones. In deployed systems, the exponent can be empirically predicted (Section 2.4.3).

2.4 Experiments with biased feedback and the evaluator

To more thoroughly understand the nature of MNCAR implicit feedback and the proposed unbiased evaluator, we studied three large-scale real-world datasets and four recommendation algorithms. Our experiments are comprised

⁴ $O_{*,i}$ satisfies the Bernoulli distribution.

of three parts: (1) investigating how popularity bias is manifested in real-world platforms, (2) exploring properties of the power-law exponent, and (3) understanding debiasing effects of the unbiased evaluator.

2.4.1 Experimental setup

To describe the setup of the experiments, we review the datasets and algorithms, describe the recommendation model implementations with OpenRec [171], and present the details of model training.

Datasets

We used three datasets of varied size and sparsity ($\frac{\# \text{ interactions}}{\# \text{ users} \times \# \text{ items}}$). For each dataset, we randomly and independently hold out 15% of user-item interactions for validation and 15% for testing, and we used the remaining 70% of records for training. During testing, we excluded cold-start users and items that have no record in the training set.

- **citeulike** [162]. citeulike is a reference management service, where scholars curate article collections based on their preferences and professional needs. We used the dataset collected by Wang et al. [162] and treated “saving an article” as a positive implicit feedback signal. The dataset contains 204,986 interactions between 5,551 users and 16,980 items (sparsity: $2e-3$).
- **Tradesy** [67]. Tradesy is a large second-hand retail market for clothing and fashion. We used the dataset released by He et al. [67], and treated “want

an item” and “bought an item” as positive signals. The final dataset includes 19,243 users, 165,906 wanted or bought items, and 394,421 interactions (sparsity: $1e-4$).

- **Amazon book** [105, 171]. The Amazon book dataset was derived from the original Amazon review dataset [105, 171]. The dataset records users’ Amazon purchasing history under the book category. The dataset covers 99,473 users, 450,166 books, and 996,938 transactions (sparsity: $2e-5$).

Algorithms

We considered recommendation models with different training procedures (pairwise and pointwise) and architectures (matrix-factorization based and metric-learning based).

- **Bayesian Personalized Ranking (BPR)** [124]. BPR is based on the general framework of matrix factorization that learns vector representations for users and items. Specifically, user u ’s preference toward item i is modeled as $\hat{x}_{u,i} = v_u^T v_i + \beta_i$, where v_* denote representations, and β_i denotes the item-specific bias. Built upon the scoring function $\hat{x}_{u,i}$, BPR trains the model parameters on (u, i, j) triplets (i and j represent interacted item and non-interacted item respectively) using a pairwise ranking based optimization framework that minimizes the following loss.

$$\min_{\Theta} \sum_{(u,i,j) \in \mathcal{D}} -\ln(\hat{x}_{u,i} - \hat{x}_{u,j}) + \lambda_{\Theta} \|\Theta\|, \quad (2.16)$$

where \mathcal{D} is the set of triplets that are *randomly* sampled from the training dataset and Θ is the set of model parameters.

- **Collaborative Metric Learning with Uniform Weights (U-CML)** [71]. U-CML is trained on the same (u, i, j) triplets as BPR, but instead of modeling user–item scores using dot products, U-CML leverages the Euclidean distance metric to regularize the embedding space, i.e., $\hat{x}_{u,i} = \beta_i - \|v_u - v_i\|^2$, where all representations are bounded within a unit sphere. Another difference between U-CML and BPR is that U-CML minimizes the pairwise hinge loss:

$$\min_{\Theta} \sum_{(u,i,j) \in \mathcal{D}} \left[m + \hat{x}_{u,i} - \hat{x}_{u,j} \right]_+ + \lambda_{\Theta} \|\Theta\|^2 \quad (2.17)$$

- **CML with Approximate-Rank Weights (A-CML)**. U-CML model randomly samples the triplets from the training set, making most of them become trivial samples as the training proceeds. Therefore, as suggested by Hsieh et al. [71], we leveraged the approximate-rank weighting technique [164] to adjust the weight of each training instance:

$$\min_{\Theta} \sum_{(u,i,j) \in \mathcal{D}} w_{u,j} \left[m + \hat{x}_{u,i} - \hat{x}_{u,j} \right]_+ + \lambda_{\Theta} \|\Theta\|^2, \quad (2.18)$$

where $w_{u,j} = \log(\text{rank}(u, j) + 1)$ and $\text{rank}(u, j)$ is the rank of item j in user u 's recommendation list. The rank can be estimated by sequential [164] or parallel [71] sampling. To speed up the training, we sampled 10 negative items in parallel for each observed user–item interaction, as suggested by Hsieh et al. [71].

- **Probabilistic Matrix Factorization (PMF)** [129]. PMF is a pointwise trained recommendation model, i.e., it is built upon pairs (u, i) . The model is optimized to minimize the following regularized square error:

$$\min_{\Theta} \sum_{u,i} c_{u,i} (r_{u,i} - \hat{x}_{u,i})^2 + \lambda_{\Theta} \|\Theta\|^2, \quad (2.19)$$

where $r_{u,i} = 1$ if user u interacted with item i , and $r_{u,i} = 0$ otherwise. Because of the sparsity of the interactions, $c_{u,i}$ is set to a higher value for

$r_{u,i} = 1$ than for $r_{u,i} = 0$. In our experiments, $c_{u,i}$ was set to 1 and 0.25, respectively, for those two cases.

Implementations and training

We implemented the algorithms based on the OpenRec framework [171]. The dimensionality of user and item representations was set to 50 for citeulike and to 100 for the other datasets. Each model was trained using the Adam optimizer [84] with a batch size of 8K. Because of differences in the sizes of the datasets, the models were trained for 50K, 120K, and 200K iterations⁵ under citeulike, tradesy, and Amazon book, respectively. We conducted *model selection* [131] for each algorithm–metric pair by training recommenders with different regularization parameters, i.e., $\lambda_\Theta \in \{0.1, 0.01, 0.001, 1e-4, 1e-5\}$. The optimal training iteration and λ_Θ value are determined by the evaluation on the validation set. The recommendation performances are finally reported on the held-out testing sets. Because of the large item space, it is computationally infeasible to compute rankings over all items. Therefore, for each user, we randomly and independently sample 200 items with which users have not interacted before and compute rankings over the sampled sets. This is a common approach adopted by recent literature [171].

⁵An iteration is defined as a feed forward and a backward propagation using a batch (size=8K) of randomly sampled training data.

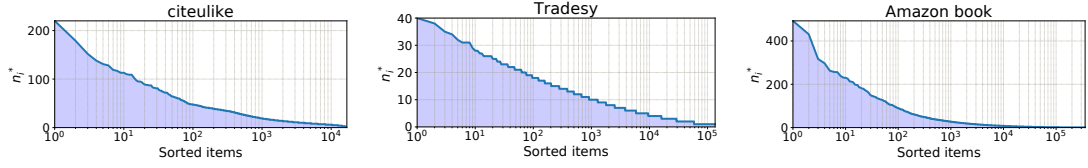


Figure 2.2: The distribution of n_i^* (the observed number of interactions with item i) in the three datasets. The items are presented in descending order of n_i^* . The horizontal axis is log scaled for better visualization. In all datasets, the n_i^* distribution is skewed and the user interactions are significantly biased.

2.4.2 Investigating popularity bias

We initially conducted an experiment to understand *to what extent popularity bias is manifested in real-world recommendation systems*. Specifically, we investigated two kinds of bias related to popularity: (1) **interaction bias** (i.e., that users tend to interact more often with popular items), and (2) **presentation bias** (i.e., that recommenders unfairly present more popular items than long-tail ones).

However, in existing datasets, interaction bias is barely separable from presentation bias [132], since a user can interact with an item only if it is presented. Therefore, we resorted to the joint effects of the two kinds of bias, which are manifested in the distribution of n_i^* , i.e., the number of times users interact with each item. Intuitively, an unbiased platform should expect users to interact broadly. As a result, user attentions are likely to be evenly distributed. On the contrary, if a platform is highly biased, then user interactions tend to be more concentrated, which leads to dominance by a small set of items. We show the n_i^* distribution for all $i \in \mathcal{I}$ in Fig. 2.2. Given that the horizontal axis is log scaled, the n_i^* distribution is significantly skewed: Most of the items received very few user interactions. For example, on Amazon book, more than 99.9% of items received fewer than 100 interactions. In addition, the degree of bias varies

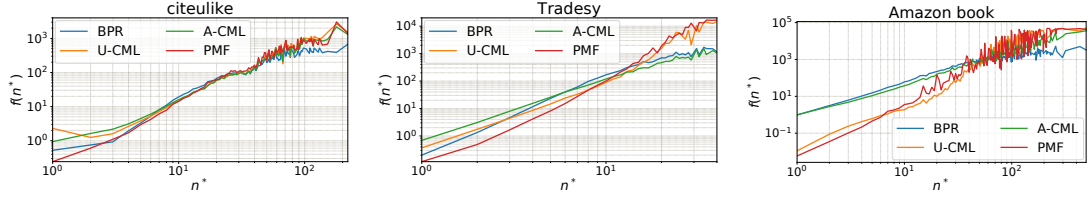


Figure 2.3: Empirically estimated $f(n^*)$ on the three datasets and the four recommendation algorithms. $f(n^*)$ denotes the average number of times that an item with observed popularity n^* was recommended. Both axes are log scaled. Therefore, exponential growth is linear in the figure. All settings manifest significant presentation bias.

across datasets: The Amazon book dataset is the most popularity biased, while the tradesy dataset is the least popularity biased.

For the presentation bias, we measured the average number of times that an item with the observed popularity $n^* \in [1, \max(n_i^*)]$ was recommended, denoted by $f(n^*)$. An unbiased system should expect a relatively flat $f(n^*)$ with a small slope, whereas a biased recommender may produce linearly or exponentially growing $f(n^*)$. We treated the top 50 recommendations that the trained recommenders made for every user as recommended items, and $f(n^*)$ was computed as follows:

$$f(n^*) = \frac{\sum_{i \in \mathcal{I}} \mathbf{1}(n_i^* = n^*) \cdot N_i}{\sum_{i \in \mathcal{I}} \mathbf{1}(n_i^* = n^*)}, \quad (2.20)$$

where N_i is the frequency of item i in all users' top 50 recommendations. For each user, the recommendation list was computed over the complete item set \mathcal{I} , excluding items that the user had already interacted with in the training set. In Fig. 2.3, we show the empirically estimated $f(n^*)$. All three $f(n^*)$ curves appear to be mostly monotonic, with small variations, which suggests that an item with small n_i^* is much less likely to be presented, compared to the ones with larger n_i^* . Also, different algorithms tend to manifest diverse patterns. For example, in

Amazon book, BPR and A-CML are more likely to present long-tail items than PMF and U-CML.

To sum up the findings, we demonstrated that both forms of popularity bias pervasively exist on platforms that use the mainstream recommendation algorithms. Although the amount of bias varies across platforms and algorithms, it appears to be highly significant. In addition, the estimation of presentation bias provides a mechanism for gaining an empirical understanding of the properties of the power-law exponent (eqn. 2.15), which is discussed next.

2.4.3 Exploring the power-law exponent

To understand the properties of γ , we estimated its value by running simulations on offline datasets. The shape of the probability distribution $\hat{P}_{*,i}^{\text{select}}$, parameterized by γ , was most likely to be affected by two factors: the recommendation algorithm (which controls *what to select*) and the content platform (which determines *what is available*). Therefore, we predict a value of γ for each algorithm–platform pair. Because $\hat{P}_{*,i}^{\text{select}}$ is determined by only an item’s observed popularity n_i^* : $\hat{P}_{*,i}^{\text{select}} \propto (n_i^*)^\gamma \propto f(n^* = n_i^*)$. Estimating the value of γ is equivalent to solving the following minimization problem:

$$\min_{\gamma} \sum_{(x,y) \in \mathcal{T}} \left(\log \left(\frac{f(y)}{f(x)} \right) - \gamma \cdot \log \left(\frac{y}{x} \right) \right)^2, \quad (2.21)$$

where $\mathcal{T} = \{(x, y) | x, y \in [1, \max(n_i^*)] \wedge x \neq y\}$. Because this is a quadratic optimization problem, γ can be analytically solved as

$$\gamma = \frac{\sum_{(x,y) \in \mathcal{T}} \log \left(\frac{f(y)}{f(x)} \right) \cdot \log \left(\frac{y}{x} \right)}{\sum_{(x,y) \in \mathcal{T}} \left(\log \left(\frac{y}{x} \right) \right)^2} \quad (2.22)$$

Table 2.2: Estimated γ value for every dataset–algorithm pair. The algorithm that achieves the lowest γ in each dataset is shown in bold-face. The γ estimation is more sensitive to the choice of datasets than to the choice of algorithms.

Dataset	BPR	U-CML	A-CML	PMF	Average
citeulike	1.67	1.64	1.55	1.89	1.69
Tradesy	2.96	2.40	2.25	3.07	2.67
Amazon book	1.85	2.11	1.70	1.80	1.87

We fit γ using the calculated $f(u^*)$ from Section 2.4.2. To make the estimation numerically more stable and robust to outliers, we exclude the top 0.5% of items that with the highest n^* . The estimated γ values are presented in Table 2.2. We find that the power-law curve accurately fits $f(n^*)$ with a small average square error. Also, among all algorithms, A-CML stands out as having the lowest estimated γ value in all three datasets, which suggests that it manifests the least presentation bias. Overall, however, the estimated γ value is relatively stable in each dataset (the range of values is 0.34, 0.82, and 0.41 for the citeulike, Tradesy, and Amazon book datasets, respectively).

These experimental results suggest that in practice, if the past recommendation algorithm is known, use of a power-law distribution can accurately fit and reconstruct $\hat{p}_{*,i}^{\text{select}}$. Even if the accurate recommender is unknown, it is still plausible to roughly predict the γ value by experimenting with classical algorithms on the given dataset. In our next experiment, we leverage the estimated γ value to understand the debiasing effects of the unbiased evaluator.

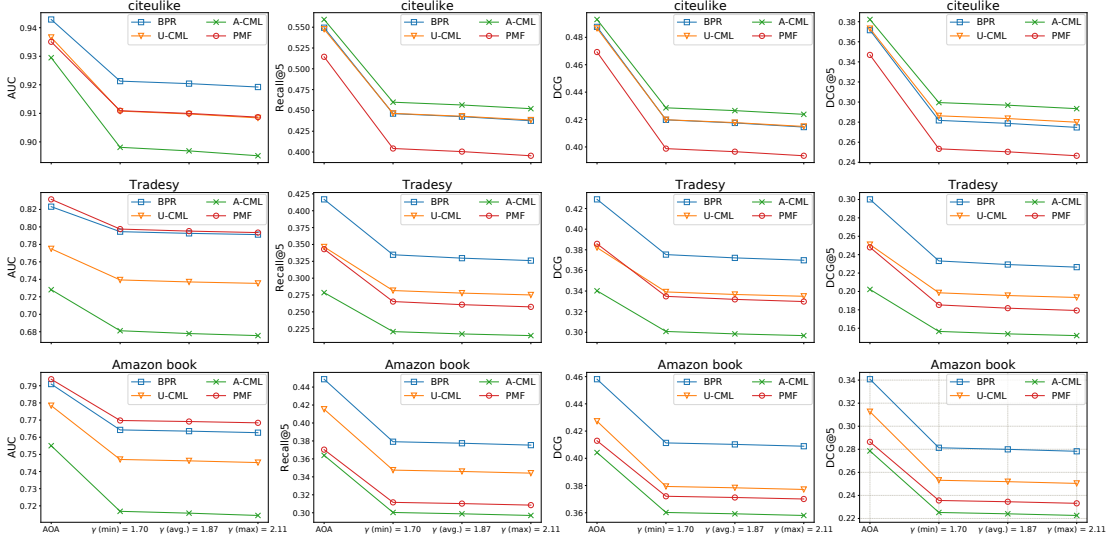


Figure 2.4: Comparison of the traditional and unbiased evaluators in measuring the performance of four recommendation algorithms. The evaluations were conducted over three datasets using four metrics. Each sub-figure represents a specific dataset–metric pair. For the unbiased evaluator, three estimated γ values from Section 2.4.3 were used in the experiments. The unbiased evaluator significantly reduces the biased weights that the AOA method places on the popular items and produces robust and consistent results for any γ from the estimated range.

2.4.4 Understanding the unbiased evaluator

We compare the outputs from the AOA and the unbiased evaluator under the same algorithm–platform settings. Specifically, for each dataset, we experiment on the *minimum*, *average*, and *maximum* γ values from Table 2.2. We evaluate models against four metrics: AUC, DCG, DCG@5, and Recall@5, as defined from eqns. 2.2 through 2.5. The experimental results are presented in Fig. 2.4. Our main findings are discussed below.

- **The unbiased evaluator reports lower performance, regardless of the algorithm, dataset, or evaluation metric.** As shown in Fig. 2.4, after apply-

ing the unbiased evaluator, the estimated recommendation performance drops significantly. This is because recommenders usually perform worse on long tail items than on popular ones, and the unbiased evaluator corrects and reduces the biased weights that AOA places on popular items. This finding reveals that *the traditional evaluation method may over-estimate the performance of recommendation algorithms*.

- **The unbiased evaluator may amplify, diminish, or flip the relative differences reported by AOA.** In many cases, the unbiased estimator does not change the absolute performance difference between algorithms, but it amplifies the relative difference, e.g., BPR outperforms PMF by 22% and 26% in terms of the Recall reported by AOA and $\gamma(\min)$, respectively. Also, the unbiased evaluator may diminish (e.g., U-CML vs. BPR under Amazon book-DCG) or flip (e.g., PMF vs. U-CML under Tradesy-DCG) the relative differences. These observations highlight a caveat that *traditional evaluation may lead to inaccurate judgements or misjudgments of the relative utility of algorithms*.
- **The outputs of the unbiased estimator are stable for different γ values from the estimated range.** In all cases, the outputs of the unbiased evaluator are stable for different γ values (min, avg., or max). In other words, as long as the γ value is from the estimated range, the unbiased evaluator is expected to produce robust evaluation results.

In summary, these results demonstrate that the unbiased evaluator is robust and has the potential to more objectively evaluate and compare different recommenders. Next, we empirically measure its debiasing performance.

2.5 Evaluating debiasing performance

We leverage the Yahoo music ratings dataset [169] to quantify the debiasing performance of the unbiased evaluator. This dataset contains users’ ratings of a uniformly randomly selected set of music, which could be used to measure recommenders’ true performances.

2.5.1 Experimental setup

The original dataset includes a training set and a testing set. The training set contains 300K ratings given by 15.4K users against 1K songs through natural interactions, whereas the testing set is collected by asking a subset of 5.4K users to rate 10 randomly selected songs. To tailor this dataset for experimenting with implicit feedback, we treat items rated greater than or equal to 4 as relevant, and others as irrelevant, as suggested in prior literature [71]. We filter the testing set by retaining users who have at least one relevant song and at least one irrelevant song in the testing set, and at least two relevant songs in the training set⁶. We hold out a biased testing set (**biased-testing**) from the training set by randomly sampling 300 songs for each user.

We train the models discussed in Section 2.4.1 using the same protocol but with fixed hyperparameters ($\lambda_\Theta = 0.001$, 10K training iterations, 50 latent factors). For each model, different evaluators are used to evaluate its performance against the biased-testing set in terms of AUC and Recall.⁷ The models’ true performances are calculated by AOA over the unbiased testing set.

⁶2,296 users satisfy these requirements.

⁷Recall@30 (biased-testing set) and Recall@1 (testing set) were compared since the biased-

Table 2.3: The Mean absolute error (MAE) between evaluators’ outputs on the biased-testing set and recommenders’ true performances. Performance was measured against AUC and Recall. For the unbiased evaluator (UB), four γ values were used in the experiments ($\gamma = 1.5, 2.0, 2.5, 3.0$).

(a) Mean absolute error (MAE) on AUC					
Model	AOA	UB(1.5)	UB(2.0)	UB(2.5)	UB(3.0)
U-CML	0.151	0.102	0.099	0.096	0.094
A-CML	0.152	0.103	0.099	0.097	0.094
BPR	0.147	0.109	0.106	0.104	0.103
PMF	0.148	0.103	0.100	0.097	0.095

(b) Mean absolute error (MAE) on Recall					
Model	AOA	UB(1.5)	UB(2.0)	UB(2.5)	UB(3.0)
U-CML	0.401	0.270	0.260	0.253	0.248
A-CML	0.399	0.274	0.264	0.258	0.253
BPR	0.380	0.275	0.268	0.262	0.258
PMF	0.386	0.267	0.259	0.252	0.248

2.5.2 Results

Table 2.3 shows the mean absolute error (MAE) between different evaluators’ outputs on the biased-testing set and the recommenders’ true performances. For both AUC and Recall, the unbiased evaluator (UB) reduces more than 30% of the errors in AOA, and UB’s debiasing performance is insensitive to the hyperparameter selections. Within the range of $[1.5, 3.0]$, UB consistently produces significantly lower errors than AOA. However, these results also demonstrate that UB has ample room for future improvements.

testing set is 10 times as large as the testing set.

2.6 Conclusions and discussions

This chapter studied the problem of evaluating ImplicitRec using biased feedback data and showed that the widely adopted AOA evaluation is biased toward popularity. Built upon the IPS technique from causal inference, we developed a theoretically grounded unbiased evaluator and empirically demonstrated its ability to significantly reduce recommender evaluation biases. However, the developed unbiased evaluator is limited in its two simplified assumptions, which points out promising future research directions. First, in the absence of detailed meta-information about users, we assumed that the propensity is user independent and that the probability of an item being presented is determined by its observed popularity. In reality, the propensity may be affected by user-specific traits and preferences. Future research could investigate more sophisticated propensity estimation methods, such as building predictive models to take auxiliary user features into consideration. And we also assumed that the probability that a user interacts with an item is independent of the probability that the item is recommended. This does not capture the potential impact of recommendations and item presentation order on users' preferences. Future research could conduct controlled user testing to model these nuanced effects.

In addition, this chapter has implications for the development of recommendation algorithms that intend to be robust to popularity bias. We showed that a recommender's accuracy on popular items usually overestimates that recommender's true performance. Algorithms that are robust to popularity bias should explore ways to improve long-tail recommendations, not only through popularity under-weighting, but also via other techniques such as stratified sampling, data augmentation, and low-shot learning.

CHAPTER 3

LEVERAGING RICHER DATA SOURCES: PERSONALIZED CREATIVE APPLICATIONS INCORPORATING UNSTRUCTURED SOFTWARE USAGE TRACES

3.1 Introduction

Debiased implicit feedback, as shown in Chapter 2, is a fruitful data source for user-centric recommendation systems. However, this data is limited when it comes to profile cold-start users whose interactions with a platform are much more sparse. To address this problem, this chapter exploits cross-platform data sources beyond implicit feedback. Specifically, we focus on modeling software usage traces to personalize creative applications [174, 175].

User actions while using software applications are recorded for the purpose of collecting application usage statistics and reproducing program errors. Such data streams are often underused by service providers. Unlike relatively clean data traces, such as text, image, and search queries, application usage records are particularly noisy and unstructured, and companies often find it hard to extract value from them. However, the social science literature [47, 94] suggests that people’s activities in professional contexts (such as using software applications) relate to their social behavior (such as online social interactions). For instance, the personalities expressed on social platforms could indicate possible software usage patterns such as proficiency and specialization. Just as social interaction traces have enabled great success in personalizing online communities, we believe that integrating application usage traces can further empower novel, effective and personalized services, as Fig. 3.1 shows.

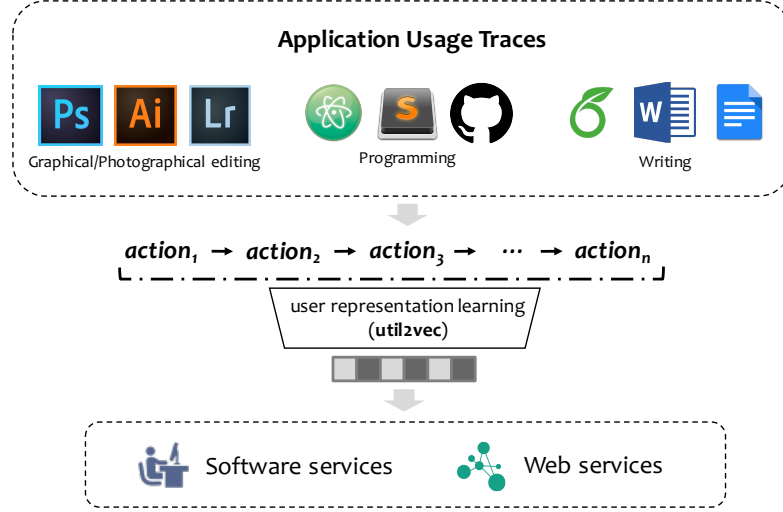


Figure 3.1: Services that can benefit from the integration of application usage traces. By leveraging the user representations learned from sequences of actions, software service providers and social platforms can improve their personalized services and empower new user experiences.

In this chapter, we explore this largely untapped space with a large number of creative professionals who use Photoshop for work and actively socialize on Behance¹, a popular large-scale online community where millions of professional photographers, designers and artists share their artwork. We demonstrate that by leveraging the data traces from shared users, Photoshop and Behance can provide significantly improved personalized services and create new user experiences. Our contributions in this chapter are summarized below.

- We develop and evaluate an approach based on distributed representation learning, **util2vec**, that produces high-quality representations of Photoshop users. This model encodes the sequence patterns of the actions each user has performed, and significantly outperforms the a *bag-of-actions* representation by 31.72% Mean Reciprocal Rank (MRR) in the *user finger-*

¹<https://www.behance.net/>

printing task. This approach can be applied to other software applications (Section 3.4).

- Based on this model, we present three sample applications:
 1. We develop and evaluate a practical tagging system for Photoshop users. The system, for the first time, is able to accurately predict areas of focus for millions of Photoshop users, who may or may not be active on Behance. Our model significantly outperforms popularity-based tagging by 31% (Recall@1), and is able to accurately predict long-tailed tags that are important but unpopular among the broader population (Section 3.5.1).
 2. We propose a two-phase recommendation method that generates more accurate recommendations for cold-start users on Behance by leveraging their previous Photoshop usage traces. The performance improvements over the popularity baseline are significant on all tested metrics including area under the ROC curve (AUC) (6.8%) and Recall@ K (21.2% when $K = 100$). Ultimately, our model enables personalized recommendations for a massive number of new users who have Photoshop usage history (Section 3.5.2).
 3. We design a novel application, named the **inspiration engine**, for Photoshop users by leveraging the co-occurrences of application usage traces and uploaded art projects on Behance. The qualitative results demonstrate how integrating these data sources enable new user experiences (Section 3.5.3).

Although the data used in this chapter comes from creative professionals, the models and frameworks studied can be applied to personalize services in

numerous similar scenarios. Especially under the evolution of app ecosystems, user activities across stand-alone software applications and social platforms can be more easily associated via proprietary or public ids, e.g., Gmail, Facebook, Creative Cloud, and Github. We believe that this opens up a new and fruitful space of future user-modeling research for private companies as well as open source communities. The technical content of the chapter is structured as follows. We introduce our dataset in Section 3.3, followed by **util2vec** model in Section 3.4. Then we present three models and applications leveraging usage traces in Section 3.5.

3.2 Related work

This chapter benefits from and has implications to multiple threads of user modeling research, and the **util2vec** model is inspired by previous work on distributed representation learning.

3.2.1 Distributed representation learning

Distributed representation learning was first introduced in the area of natural language processing [128]. The goal is to learn a vector space for all words so that they can be used as inputs to natural language understanding algorithms [108]. Recently, such an approach has been extended and successfully applied to paragraphs [93], medicine [29] and online purchases [60]. For instance, Grbovic et al. [29] proposed a framework to learn a vector representation for each product and user given the historical purchasing records, and Choi et

al. [29] demonstrated that a similar approach can be applied to learn hierarchical representations for medical concepts. Our **util2vec** framework is inspired by the previous research efforts mentioned above, and to the best of our knowledge, this is the first work to design a distributed representation learning algorithm in the domain of software user modeling.

3.2.2 User modeling in online social platforms

For online social platforms, personalization and user modeling are important tasks, since appropriately matching customers and products is a key to satisfactory user experiences [90]. Often, the goal of such modeling is to derive a real-valued vector for each user that summarizes his/her preferences, habits, and traits in online social platforms. Previous work constructs user vectors by leveraging intra-platform interactions [149], e.g., ratings [90, 13], purchases [66], content consumption [6, 158, 73], reviews [183], and social networks [63, 62], or cross-platform interactions, e.g., personal data streams across email, Twitter, and Facebook [72], and follower-followee connections across YouTube and Twitter [170]. Learned user representations have been shown to be effective in many application domains, such as movie [13], television [73], article [72, 161], e-commerce [66] and social network [62, 63, 6].

3.2.3 Software user and command modeling

Modeling software users' proficiency based on the actions they perform has been previously studied in the context of command-recommendation sys-

tems [104, 97, 40]. The goal of such a recommender is to help users learn commands in a complex software application. However, the user modeling under such a circumstance is limited to a specific application because of the narrow scope that the modeling system is exposed to. In this chapter, we show that by integrating application usage traces with online social interactions, the potential applications that such data traces can empower are much broader and diverse. Specifically, we demonstrate that the Photoshop service provider can conduct better user tagging and create new user experiences.

Another line of related work around application usage records attempts to understand the semantic meanings of software actions [3, 52]. By training a word2vec model [108] on online documents, previous work [3] discovered correspondences and relationships between natural language and software actions, which was used to fuel tutorial-recommender systems. Although our model is not directly optimized for this task, we can still extract semantic meanings of actions and their relationships to users' social interactions, because the actions, along with the users, are embedded in the same feature space (Section 3.4).

Although previous research on user modeling has achieved great success, most models only consider data from within the online social platforms. In this chapter, we demonstrate that by leveraging users' digital traces from application usage records, online social platforms can better understand users and provide more effective recommendations.

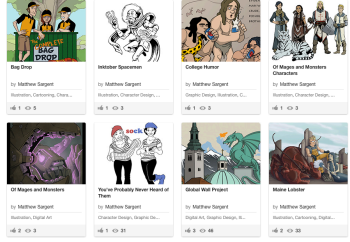
3.3 Dataset

We associate action histories from Photoshop with social interactions on Behance through Creative Cloud accounts as people use them to log into both services. The reasons why we choose these two platforms are three-folded. (1) Photoshop is one of the most popular computer software applications used by creative professionals, and it is an indispensable daily component for people across many creative occupations including graphical designer, photographer, and architect. Therefore, it is an ideal context in which to study and impact users' working behavior at a large scale. (2) Behance possesses an abundant user base as millions of creative professionals share their work and socialize with each other on the platform. Also, it is one of the major websites for creative talent search. (3) As Photoshop and Behance both serve creative professionals, there are many shared users for us to investigate.

In Photoshop, all of the actions performed in the application, e.g., buttons clicked and features applied, are collected from the users who enabled application usage reporting. An example of the action sequence is shown in Fig. 3.2. We target a group of users from the U.S. and their action histories from January 2015 to June 2015. We selected **22 billion** actions from **3 million** unique Photoshop users. From the Behance platform, we collected users' social interactions in three categories: (1) self-disclosed areas of focus, e.g., *Cartooning*, *Interaction Design* and *Fashion*; (2) user-uploaded projects; and (3) users' view and appreciate history on these projects. An example of the collected information from Behance is shown in Fig. 3.2. In this chapter, we select 0.86 million behance users where 67 thousand of them are also among the Photoshop users mentioned above.

[UID]	[SID]	2015-03-09 22:22:16	Open
[UID]	[SID]	2015-03-09 22:23:07	New_Slice
[UID]	[SID]	2015-03-09 22:23:15	Resize_Slices
[UID]	[SID]	2015-03-09 22:24:06	New_Guide
[UID]	[SID]	2015-03-09 22:24:12	New_Guide
[UID]	[SID]	2015-03-09 22:24:40	Copy_Slice
[UID]	[SID]	2015-03-09 22:24:47	Drag_Slice
[UID]	[SID]	2015-03-09 22:24:51	New_Guide
[UID]	[SID]	2015-03-09 22:25:00	Copy_Slice
[UID]	[SID]	2015-03-09 22:25:06	Drag_Slice
[UID]	[SID]	2015-03-09 22:25:14	Copy_Slice
[UID]	[SID]	2015-03-09 22:25:22	Drag_Slice
[UID]	[SID]	2015-03-09 22:25:23	Delete_Slice
[UID]	[SID]	2015-03-09 22:25:27	Copy_Slice
[UID]	[SID]	2015-03-09 22:25:30	Drag_Slice
...			
*UID: User ID *SID: Session ID			

(a) An example of action sequence in Photoshop

projects viewed	
PROJECT VIEWS	127
APPRECIATIONS	12
FOLLOWERS	3
FOLLOWING	7
areas of focus	
FOCUS	
Illustration, Character Design, Digital Art	
uploaded projects	
	

(b) An example of the collected user's social interactions on Behance

Figure 3.2: Data samples of Photoshop usage records (left) and social interactions on Behance (right). We collected three categories of social interactions for each user: *projects viewed*, *self-disclosed areas of focus* and *uploaded projects*.

3.4 Software user representation

In this section, we propose an accurate and robust user modeling framework to model the action histories of software users. We start by introducing the model, followed by implementation details and performance evaluations.

3.4.1 util2vec framework

Given the action history $\mathbf{H}_u = (a_1^u, a_2^u, \dots, a_n^u)$ from a software user u , our goal is to learn a fixed-length real-valued vector \mathbf{v}_u that represents his/her software usage

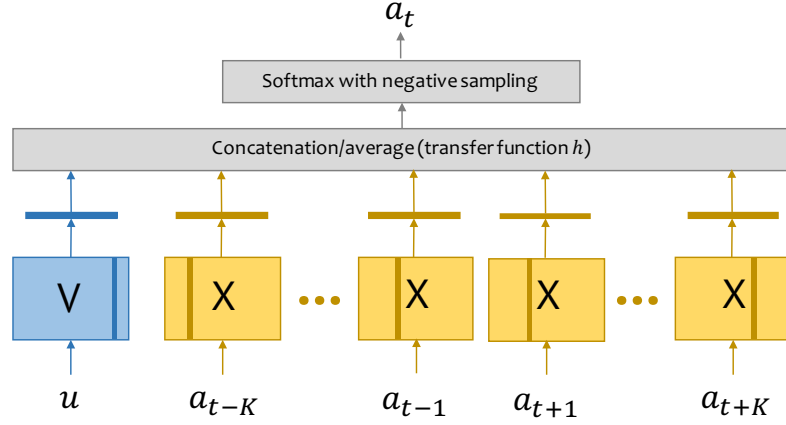


Figure 3.3: The architecture of util2vec model. The columns of V and X store the user representations and action representations respectively. While the action embedding X is shared across different users, user embedding V is user-specific.

pattern. We propose a framework named **util2vec** to learn the user representation. In our framework, each user or action is mapped to an M -dimensional vector, and the vectors are trained to maximize the log probability, as defined in eqn. 3.1, across all users.

$$\frac{1}{T - 2K} \sum_u \sum_{t=K}^{T-K} \log p(a_t^u | a_{t-K}^u, \dots, a_{t+K}^u \setminus a_t^u), \quad (3.1)$$

where T is the total number of actions from a given user, and K is the farthest action before/after the prediction target that is used as the context. In other words, the size of the sliding window is $2K + 1$. Intuitively, the model optimized for the objective defined in eqn. 3.1 will be able to predict any action given the context of the user and the surrounding actions.

For the prediction, we use the softmax function to model the conditional probability $p(a_t | a_{t-K}, \dots, a_{t+K} \setminus a_t)$ as follows (We omit the superscripts of a_t^u where they are clear from context).

$$p(a_t | a_{t-K}, \dots, a_{t+K} \setminus a_t) = \frac{e^{y_{a_t}}}{\sum_i e^{y_i}}, \quad (3.2)$$

where the vector $y = b + Wh(u, a_{t-K}, \dots, a_{t+K} \setminus a_t; V, X)$; the bias vector b and weight matrix W are parameters of the model, and the columns of the matrices V and X store the user and action representations respectively, i.e., $v_u = V[:, u]$ and $x_i = X[:, i]$ in *numpy*-style notation. The parameters b, W, V , and X are learned during training. In the **util2vec** framework, we use a transfer function h that averages or concatenates a user representation with representations from $2K$ context actions, as Fig. 3.3 shows.

We use Stochastic Gradient Descent (SGD) to conduct the training. The model is trained with action histories from U unique Photoshop users ($U = 3$ million), and the user and action representations are updated concurrently. After the model is trained, we can infer a new user u 's representation v_u by fixing the parameters b, W, X and only fitting the vector v_u to user u 's action history.

3.4.2 Implementation details

Along with **util2vec**, we use negative sampling and additional action preprocessing steps to speed-up the training and reduce the noise, which will be discussed next.

Negative sampling

It is expensive to compute the softmax function in eqn. 3.2, since the denominator involves a sum over a large number of unique actions. To avoid this cost, we replace the softmax loss with a negative-sampling loss. This strategy has been successfully applied in the word2vec model [108]. Specifically, for each

instance, we randomly sample S actions that are different from the target action a_t and approximate the log probability $\log p(a_t|a_{t-K}, \dots, a_{t+K} \setminus a_t)$ as follows:

$$\log(\sigma(y_{a_t})) + \sum_{s \in S} \log(\sigma(-y_s)), \quad (3.3)$$

where S is a set of randomly sampled actions such that $a_t \notin S$, and σ is the sigmoid function $\sigma(x) \equiv \frac{1}{1+e^{-x}}$.

Preprocessing and parameter settings

Preprocessing: For each action, we keep it in the vocabulary only if it is used by at least 100 unique users, and the final size of the vocabulary is 1990. During the preprocessing, we also add a special separation token [E] between two sessions to indicate the boundary of action sequences.

Parameter settings: the hyper-parameters of our model are set as follow: (1) The dimensionality of the representations, M , is set to 500. (2) The sampling window size, $2K + 1$, is set to 11, i.e., $K = 5$. During training, we use 0.025 as the initial learning rate and subtract it by 0.005 for each subsequent epoch (5 epochs in total). For inference, we use 0.1 as the initial learning rate and subtract it by 0.02 for each subsequent epoch (5 epochs in total). Our parameter settings are consistent with the previous work on word2vec [108], although further tuning might yield better performance.

3.4.3 User profiling performance

We evaluate the profiling performance of the **util2vec** model with a *user fingerprinting* task. We start by holding out the 200 most recent sessions from Photo-

shop users who have at least 400 sessions in the first 6 months of 2015 (In total, 15,369 unique users are selected). We then train the **util2vec** model over the rest of the action sequences from 3 million Photoshop users. For each of 15,369 users, her action history \mathbf{H}_i has been divided into training sub-sequence, i.e., $\mathbf{H}_i^{\text{train}} = \mathbf{H}_i[: -200]$ and validation sub-sequence, i.e., $\mathbf{H}_i^{\text{val}} = \mathbf{H}_i[-200 :]$, and an ideal model should be able to link $\mathbf{H}_i^{\text{val}}$ with $\mathbf{H}_i^{\text{train}}$ based on generated profiles. We infer the user’s representation based on the two subsequences respectively, i.e., infer $\mathbf{v}_i^{\text{train}}$ from $\mathbf{H}_i^{\text{train}}$ and $\mathbf{v}_i^{\text{val}}$ from $\mathbf{H}_i^{\text{val}}$. For each user i and her profile $\mathbf{v}_i^{\text{train}}$, we predict which validation subsequence belongs to her using cosine similarities. More specifically, we sort all the validation subsequences $\mathbf{H}_j^{\text{val}}$ by the similarities between $\mathbf{v}_j^{\text{val}}$ and $\mathbf{v}_i^{\text{train}}$ in a descending order, and the ranking of the user’s real validation subsequence $\mathbf{H}_i^{\text{val}}$ is denoted as rank_i . Finally, Mean Reciprocal Rank (MRR), as defined in eqn. 3.4, is used to evaluate the overall fingerprinting accuracy across N users ($N = 15369$).

$$\text{MRR} = \frac{1}{N} \sum_{i=1}^N \frac{1}{\text{rank}_i} \quad (3.4)$$

We compare **util2vec** to the *bag-of-actions* model, which counts the frequency with which each action occurred. As shown in Table. 3.1, our framework outperforms the baselines by 31.72% even when tf-idf is leveraged to down-weight the frequent actions. The experimental results demonstrate that our model is able to produce user vectors that are more representative and have stronger discriminative power. Generally speaking, for a given user, our representation is able to discriminate against 31.72% more distractors, and in practice, software service providers can use such representations to better fingerprint each user.

Along with the user representations, **util2vec** also learns an action embedding X that encodes semantic similarities between actions. For example, we

Table 3.1: User fingerprinting performance (hold-out session retrieval) in terms of mean reciprocal rank (MRR). The improvement is relative to the *bag-of-actions+tf-idf*.

Modeling framework	MRR (\pm standard error of mean)
util2vec	0.8238\pm 0.0029
bag-of-actions + tf-idf (baseline)	0.6037 \pm 0.0037
bag-of-actions	0.5944 \pm 0.0037
% of improvement	31.72%

present the nearest neighbors of five Photoshop actions in Table. 3.2 (the neighbors are ranked by the cosine similarities between action embeddings in descending order), and the retrieval results show that the actions are grouped by their functionalities and usage affinities. The action embeddings may also be useful for the service improvements as it tells the common software usage patterns among the population. Given the scope of this chapter, we leave further investigation as future work.

3.5 Applications

In this section, we build and present three applications that can benefit from the integration of such usage traces: *software user tagging*, *cold-start art project recommendation* and *inspiration engine*.

Table 3.2: 5-nearest neighbors of selected actions according to the action embedding X . The actions in the first row are the index, and the five actions below are the corresponding nearest neighbors. (From left to right, the actions are related to *video editing*, *font awesome icons*, *blur filters*, *path manipulations* and *shadow effects*, respectively.)

modify_video_clip	fa_times	delete_smart_filter_blur	drag_path	inner_shadow
modify_video_clip_audio	fa_user	edit_filter_effect_blur	duplicate_paths	inner_glow
set_work_area_start	faBars	edit_filter_blending_options_blur	nudge_paths	bevel_emboss
add_audio_clips	fa_home	delete_smart_filter_blur_more	scale_paths	gradient_overlay
mute_audio_track	fa_map_marker	delete_smart_filter_gaussian_blur	drag_anchor_points	clear_effects
modify_audio_clip	fa_plus	enable_filter_effect_blur	distribute_horizontal_centers	drop_shadow

3.5.1 Software user tagging

User tagging is an important task for software service providers as accurate tags are fundamental to effective business and ads targeting, personalization and recommendation. Essentially, the goal of user tagging is to assign a set of relevant tags to users based on her behavior in the platform. For Photoshop, in particular, the tagging task is to predict users' areas of focus based on the software usage patterns. For example, an ideal tagging system should be able to predict whether a user is focusing on *web design*, *UI/UX* or *architecture* based on the tools that she uses. Traditionally, building such a user tagging system requires a significant amount of domain knowledge and human labor to bootstrap the training labels. The expert software developers need to manually examine the raw usage histories and come up with the tags for a large number users. As imagined, such a human labeling process subjects to diverse human expertise in recognizing the patterns and is inevitably error-prone and incomprehensive.

We build an accurate Photoshop user tagging system with minimal human efforts by leveraging interactions on Behance. It is based on the observation that nowadays, people self-disclose their expertise and areas of focus in many social platforms for socializing and job hunting. By leveraging the self-disclosed tags from Behance and the accurate user representations derived from **util2vec**, we are able to build a user tagging system that is more accurate and robust than other approaches.

User tagging model

Formally speaking, given U users, along with their self-disclosed tags from Behance, \mathbf{t}_u and representations \mathbf{v}_u derived from Photoshop usage traces (\mathbf{t}_u is a D -dimensional one-hot encoded vector, where D is the size of the tag set), we learn a user tagging model f that takes \mathbf{v}_u as input and produces an output to approximate \mathbf{t}_u . As suggested in the image tagging tasks, we train the user tagging model by minimizing the following sigmoid cross-entropy loss:

$$-\frac{1}{U} \sum_u \mathbf{t}_u \log(\sigma(f(\mathbf{v}_u))) + (1 - \mathbf{t}_u) \log(1 - \sigma(f(\mathbf{v}_u))), \quad (3.5)$$

where the value of the j -th element in \mathbf{t}_u , $\mathbf{t}_u[j]$, is 1 if the j -th tag is selected by user u , 0 otherwise. Theoretically, model f can be any linear or non-linear function. In this thesis, we use the linear projection, i.e., $f(\mathbf{v}_u) = b + W\mathbf{v}_u$, though adding non-linear components such as multi-layer perceptron (a.k.a. deep neural networks) might potentially improve the performance.

We train the model using limited-memory BFGS (l-bfgs) [100] over the areas that are indicated by at least 100 active users on Behance. With such a filtering criteria, we finally keep 67 labels in the tag pool, which includes, to name but a few, *Graphic Design*, *Motion Graphics*, *Character Design*, *Cinematography*, *Icon Design* and *Computer Animation*. Then for any Photoshop user (without any requirement to be on Behance), we can assign tags by running the classifier over her application usage history.

Evaluation and analysis

To demonstrate the effectiveness of our tagging system, we conduct an evaluation against 65,331 users who have labeled themselves with at least one of the

67 tags. In the final dataset, each user is associated with 1 to 5 tags. We randomly divide the users into a training set and a validation set, which consists of 45,331 and 20,000 samples respectively. The baseline approach that we compare to ranks the tags purely based on their number of appearances on the Behance platform. While simple, such a comparison directly reflects the feasibility and reliability of the tagging system, and it is the best we can achieve without our tagging model. We use the average recall rate, Recall@K , as defined below, to quantitatively compare the tagging performance.

$$\text{Recall@K} = \frac{\text{number of correct tags in top } K \text{ predictions}}{\text{total number of tags in the ground truth set}} \quad (3.6)$$

The results in Table. 3.3 show that the models leveraging software usage history significantly outperform the baseline that is agnostic to such information, and the improvements are particularly remarkable for top-ranked tags—the system achieves 31.0% and 35.0% improvements in terms of Recall@1 and Recall@2 respectively. This justifies that, practically, our system can not only predict tags that are popular, but the ones that are long-tailed. Ultimately, our tagging system can make accurate predictions for millions of Photoshop users, who may or may not be active on Behance, and it is valuable to enable customized business for the service provider.

Qualitatively, we show the outputs of two tagging approaches for 6 representative Photoshop users in Fig. 3.4. For each user, we present the ground truth tags, the tags predicted by our system and the popular tags. In addition, we include user’s Behance portfolio (uploaded projects) side-by-side for the illustration purpose. But this information is not available to the tagging algorithm

Table 3.3: User tagging performance in terms of Recall@K. We use boldface for the best performed approach and feature set. The percentage of improvements are the comparison between util2vec (boldface) and popular tags. Our tagging system outperforms the popularity tags baseline by 31.0% and 35.0% in terms of Recall@1 and Recall@2 respectively.

Recall@K		1	2	3	4	5
tagging with software usage data	util2vec features (500 dim)	0.2320	0.3569	0.4466	0.5140	0.5691
	bag-of-actions+tf-idf features (1990 dim)	0.2246	0.3489	0.4352	0.5017	0.5559
tagging without software usage data	popular tags (baseline)	0.1771	0.2644	0.3644	0.4309	0.4781
% of improvements		31.0%	35.0%	22.6%	19.3%	19.0%

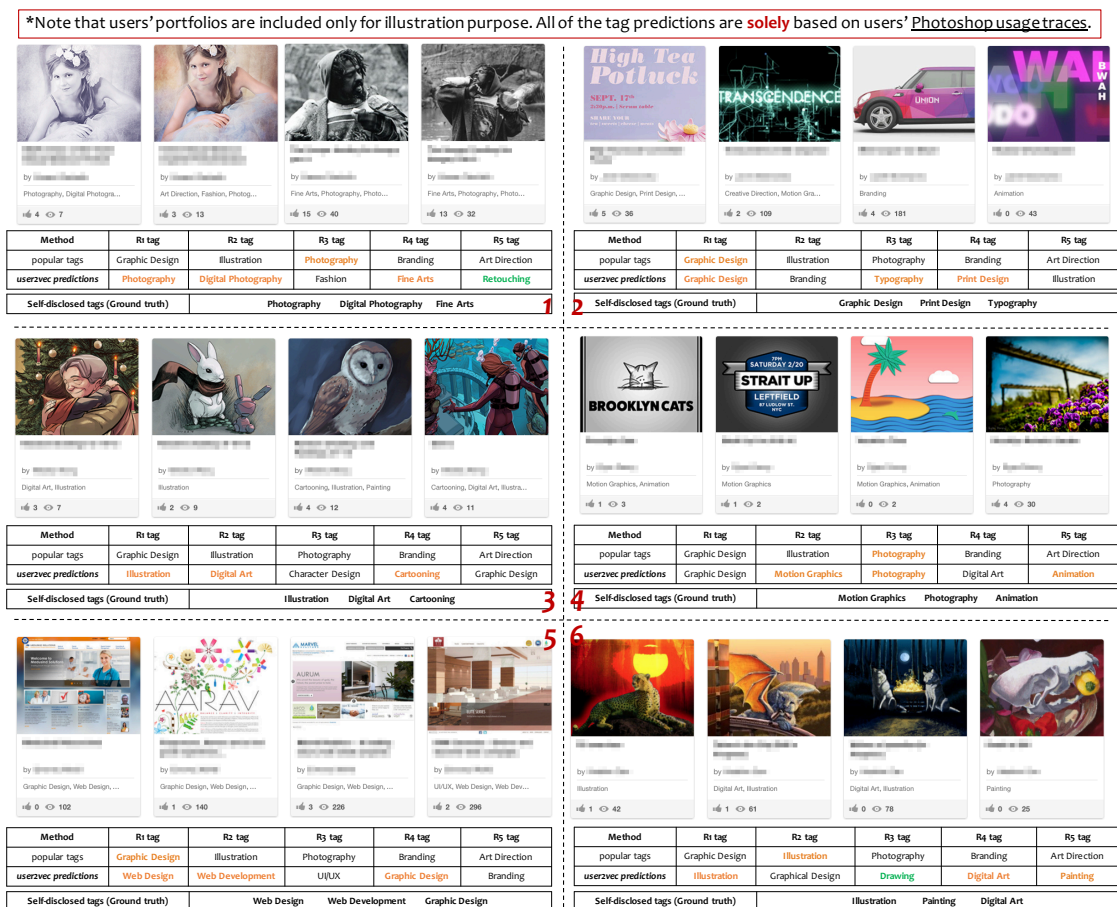


Figure 3.4: Six user tagging examples with two different approaches. For each user, we show her portfolio, top 5 tag predictions with util2vec feature, top 5 most popular tags and self-disclosed tags. The tags with orange color are the correct predictions, and the ones with green color are the ones that are inferrable from the portfolio but not explicitly selected by the user.

under any circumstance². From Fig. 3.4, we find that our tagging model is especially advantageous in the following aspects.

- **Tag diversity.** We can accurately predict a diverse array of areas of focus based on the Photoshop usage traces, e.g., *Photography* (U1, U4), *Fine Arts*

²The tagging model is mainly designed to classify Photoshop users who do not have Behance profile.

(U1), *Web Design* (U5), *Typography* (U2), *Cartooning* (U3), *Animation* (U4), *Painting* (U6), etc. The tags can be popular on the platform, e.g., *Graphic Design* and *Photography*, or long-tailed (infrequent), e.g., *Motion Graphics*, *Cartooning*, *Painting*, etc. The prediction results justify the robustness of our system when it is applied to diverse application usage patterns.

- **Generalization power.** Although there is a high correlation between user tags and appearances of uploaded art projects, as shown in Fig. 3.4, some users didn't exhaustively select all of the tags that are related. This limitation is partially addressed by the generalization power of our linear classifier. For example, based on U1's portfolio (images with same content but different coloring), there is a high chance that she is focusing on *retouching*, which is not selected by herself. Nevertheless, our system can still make reasonable predictions that include *retouching* in the top tags. This characteristic is further verified in the U6 example (tag *Drawing*).

Overall, we have shown that by modeling application usage traces, we are able to build an accurate and practical user tagging system for Photoshop with minimal human effort.

3.5.2 Cold-start art project recommendation

Cold-start is a well-known hard problem in the design of modern recommender systems. Specifically, *user-cold-start* [72] refers to the scenario where the recommendations are targeting new users, and *item-cold-start* [67] describes the case when a new item needs to be included in the recommendation pool. In *user-cold-start*, since we lack the information of her activities within a platform, a typical

solution is to either recommend the most popular items, which is not personalized, or leverage side information, such as gender, age [117] and personal data traces [72]. However, in many cases, when a new user shows up in online social platforms, their application usage records are already available. If we could leverage these data traces and properly use them to inform the recommender, there is a great potential for the social platforms to improve their cold-start recommendations. For example, Behance might be able to generate better recommendations for **3 million** Photoshop users, which is almost **4 times** the current number of Behance users. In this section, we propose a two-phase recommendation framework that leverages Photoshop usage data in recommending artistic projects on Behance.

Two-phase recommendation framework

Our recommendation framework is inspired by previous research on content-based music recommendation [158] that incorporates audio features in solving *item-cold-start* problem. We take advantage of the opportunity that a portion of Photoshop users are already active on Behance and have left a significant amount of implicit feedback, e.g., project views. Therefore, to build the recommendation pipeline, we first learn users' and items' latent factors from their project views and then build a function to map the application usage features extracted from *util2vec* or *bag-of-actions*, to the latent factors. In production, for any Photoshop user, the system conducts cold-start recommendations by first predicting user's latent factors based on her application usage data, and then ranking the items accordingly in the latent space. Formally speaking, we build the cold-start recommendation system with the following two steps (Fig. 3.5).

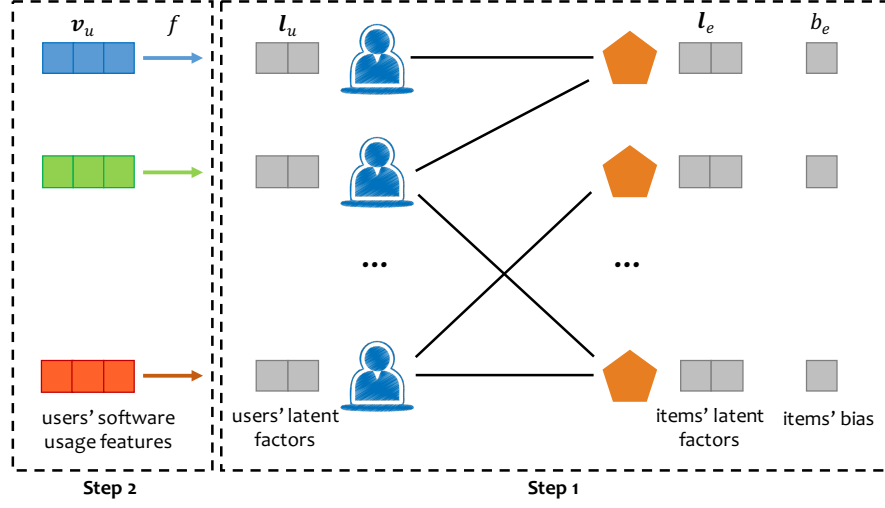


Figure 3.5: Two-phase recommendation framework. In step 1, we derive users' latent factors and items' latent factors and bias from their implicit feedback (project views). In step 2, we learn a projection function f to map software usage features to the corresponding users' latent factors.

Step 1. The goal of the first step is to learn each user u 's latent factors \mathbf{l}_u , and each item e 's latent factors \mathbf{l}_e and bias b_e , such that the value of r_{ue} , which is defined as $r_{ue} = \mathbf{l}_u^T \mathbf{l}_e + b_e$, is proportional to user u 's preference level towards item e . We learn the parameters by leveraging users' project views on the platform. Considering that such signals are implicit feedback, as suggested by [164], we propose to minimize the following Weighted Approximately Ranked Pairwise (WARP) loss:

$$\sum_{u, e \in P_u, e' \in S \setminus P_u} \ln\left(\frac{Y}{M_{e'}}\right) |1 - \mathbf{l}_u^T \mathbf{l}_e - b_e + \mathbf{l}_u^T \mathbf{l}_{e'} + b_{e'}|_+, \quad (3.7)$$

where S denotes the set of all items, P_u denotes the set of items viewed by user u , Y denotes the total number of items, and $M_{e'}$ denotes the number of negative sampling conducted before encountering an item e' that produces non-zero loss. In other words, during training, for each item that the user viewed, we keep sampling negative items e' until $1 + \mathbf{l}_u^T \mathbf{l}_{e'} + b_{e'} > \mathbf{l}_u^T \mathbf{l}_e + b_e$ is satisfied.

Step 2. In the second step, we learn a projection function f that takes a user’s software usage feature \mathbf{v}_u as input and produces output $f(\mathbf{v}_u)$ to approximate her latent factors \mathbf{l}_u . We propose to minimize the l_2 loss $\sum_u \|f(\mathbf{v}_u) - \mathbf{l}_u\|^2$ for regression.

In this thesis, we use linear function f , i.e., $f(\mathbf{v}_u) = b + W\mathbf{v}_u$. However, any non-linear function should be directly applicable here, and we leave it as future work. During training, for **step 1**, the parameters are learned with mini-batch Adagrad [39], the dimensionality of the latent factors and learning rate are set to 50 and 0.05 respectively. For **step 2**, we use l-bfgs to find the optimal solution since the optimization target is convex.

In practice, for any cold-start user u and her Photoshop usage feature \mathbf{v}_u , the items’ recommendation rankings are based on the value of $r_{ue} = f(\mathbf{v}_u)^T \mathbf{l}_e + b_e$ where the item with higher value of r_{ue} will be recommended earlier.

Evaluation and analysis

We evaluate the performance of our cold-start recommendation system by holding out a validation set from the view histories of 67,805 users. We randomly sample 10,000 users among the people who have viewed at least one project after July 1st, 2015 and regard them as the cold-start users. The most recent viewed project e_{p_u} from each cold-start user u is then held for validation, and the rest 57,805 users’ complete view histories are used for training. All the items appear in the training set are included in the items pool, which yields **5.8 million** candidates for recommendation. The time restriction is used to guarantee the causality of recommendation as the Photoshop usage data is collected from the first 6 months of 2015. During the validation, for each cold-start user, we

only use her software usage data to make the preference prediction, without relying on any previous views. Therefore, our evaluation results can properly reflect the system performance when serving cold-start users in practice.

We compare our recommender to the baseline algorithm that ranks the items based on their popularity (total number of views received). This is shown to be a very strong baseline for the cold-start recommendations [72]. Similar to user tagging, we use Recall@ K defined in eqn. 3.8 and area under the ROC curve (AUC) defined in eqn. 3.9 to evaluate the recommendation performance (N=10,000).

$$\text{Recall@}K = \frac{1}{U} \sum_{u=1}^U \delta(e_{p_u} \text{ in the top } K \text{ items for } u) \quad (3.8)$$

$$\text{AUC} = \frac{1}{U} \sum_{u=1}^U \frac{\sum_{e'} \delta(f(\mathbf{v}_u)^T \mathbf{l}_{e_{p_u}} + b_{e_{p_u}} > f(\mathbf{v}_u)^T \mathbf{l}_{e'} + b_{e'})}{\text{size of the items pool}} \quad (3.9)$$

The experimental results shown in Table. 3.4 demonstrate that all of the recommenders that leverage the Photoshop usage traces and two-phase recommendation framework perform significantly better than the baseline in terms of Recall@ K and AUC. For top-ranked items (Recall@100), in particular, our recommender outperforms the popularity based recommendation by 21.2%, which means that the users will potentially appreciate 21.2% more items among which we recommend. Also, the performance improvement suggests that we are able to personalize item recommendations to creative professionals who are new to the Behance platform.

Table 3.4: Art project recommendation performance for cold-start users in terms of Recall@K and AUC. We use boldface for the best performed approach and feature set. The percentage of improvements are the comparison between the approach in boldface and the baseline method (popular items).

Recall@K		100	200	300	400	500	AUC
cold-start recommendation with software usage data	util2vec features (500 dim)	0.0143	0.0213	0.0261	0.0313	0.0356	0.8202
	bag-of-actions+tf-idf features (1990 dim)	0.0138	0.0209	0.0269	0.0309	0.0350	0.8166
cold-start recommendation without software usage data	popular items (baseline)	0.0118	0.0188	0.0218	0.0281	0.0297	0.7683
	% of improvements	21.2%	13.3%	23.4%	11.4%	19.9%	6.8%

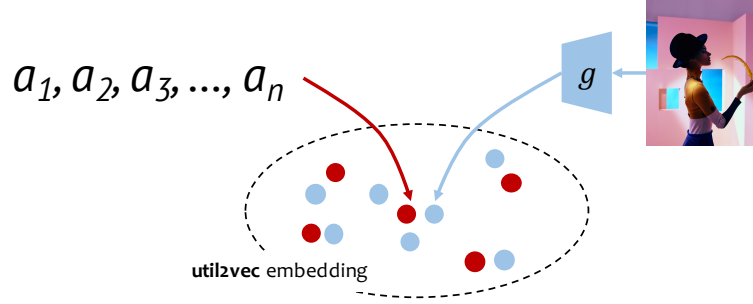


Figure 3.6: The algorithm framework for the inspiration engine. We learn a function g to project image features to the util2vec embedding space such that true actions-image pairs are close to each other and false pairs are farther away.

3.5.3 Inspiration engine

In this section, through a sample application named **inspiration engine**, we demonstrate that the data integration can also enable innovative user experiences. The goal of inspiration engine is to provide real-time and personalized inspirations for creative professionals when they are working in Photoshop, and the system is able to show the potential outcomes of the actions that have been or are likely to be performed. Such presentations are inspiring because the artists can explore a wider range of possibilities that are related but different from their current work.

Technically speaking, the core component of such application is a search engine that can return art projects that are likely to be produced by a given sequence of Photoshop actions. We build the system by leveraging the weak correspondence between the pairs of users' Photoshop usage traces and the projects that they uploaded to Behance. With such pairs, we can learn a heterogeneous joint embedding where the true actions-image pairs are close to each other, and the false pairs are further away. As shown in Fig. 3.6, for each actions-image

Table 3.5: Action-image retrieval performance in terms of Recall@K. We use boldface for the best performed approach.

Recall@K	100	300	500	AUC
inspiration engine (util2vec features)	0.0244	0.0603	0.0884	0.6646
inspiration engine (bag-of-actions+tfidf)	0.0181	0.0488	0.0741	0.6357
random guess	0.005	0.015	0.025	0.5

pair $((a_1^i, a_2^i, \dots, a_n^i), c_i), i = 1, 2, \dots, n$, we first extract features for the action sequence and the image respectively, denoted as \mathbf{v}_i and \mathbf{z}_i . In our prototype, we extract \mathbf{v}_i from **util2vec** and \mathbf{z}_i from the pooling layer (2048 dim) of pre-trained ResNet [65], the state-of-the-art image feature extractor. Then we learn a function g to project image features \mathbf{z}_i to the **util2vec** embedding space such that the objective $\sum_i \|\mathbf{v}_i - g(\mathbf{z}_i)\|^2$ is minimized.

To prototype the system, we train a linear projection function g with 353,205 actions-image pairs from 43,441 users and validate it over 20,000 held-out pairs from 20,000 users, i.e., each user contributes exactly one pair in the validation. There is no user overlap in the training and validation set, and the training is conducted using l-bfgs algorithm. Quantitatively, we use Recall@K and AUC as defined in Section 3.5.2 to evaluate the system performance, and the results are shown in Table. 3.5. The improvements over the random guess baseline justify that there is a close relationship between the Photoshop usage pattern and visual appearance of art project. In addition, in Fig. 3.7, we show four qualitative retrieval results of the inspiration engine (we only show retrieval with a single action, but our technique is applicable to action sequence as well). The nearest









































action	Top 10 nearest neighbors in the util2vec embedding space									
fade_smart_blur										
drag_path										
enable_filter_effect_lighting_effects										
rotate_canvas										

Figure 3.7: Four image retrieval results of the inspiration engine using single action. The retrieval results reflect the context where each action is often used. For example, with *fade_smart_blur*, returned images have blurred background and fading effects, and with *rotate_canvas*, images tend to have repetitive patterns.

neighbors of each action reflect the scenarios where it is often used. For example, the action *drag_path* is heavily used in web design, and the *rotate_canvas* is typically leveraged to create repetitive patterns. We will conduct an end-to-end further user study in the future to evaluate the engine.

Through three applications, we observe that the improvements brought by **util2vec**, compared to the *bag-of-actions+tfidf* model, are contingent on the context of end applications. Nevertheless, the performance improvements are significant under most of the metrics except Recall@300 in the cold-start recommendation task, so we can safely conclude that **util2vec** is beneficial in modeling unstructured application usage traces, and we may get further gains in the future by tuning the model parameters and training methods.

3.6 Conclusions

This chapter personalized software and web applications for creative professionals by leveraging Photoshop usage traces. These data enhanced existing services provided by Photoshop (i.e., accurate prediction of users' areas of focus, Section 3.5.1), and Behance (i.e., personalized recommendation for cold-start users, Section 3.5.2), and enabled new experiences (i.e., inspiration engine, Section 3.5.3) for millions of users. Our followup work [175] demonstrates that they can also be used to characterize user skills.

Although we mainly focused on the platforms for creative professionals, our results suggest that such an integration is a fruitful source for future personalization research, and can potentially have a great impact on a larger population. For example, personalized applications can be built for programmers based on their Github usage records, and for journalists based on the usage of document editing tools. As people's work and leisure lives are increasingly accompanied by applications, leveraging digital breadcrumbs that they left behind is crucial to achieving user-centric recommendation systems.

CHAPTER 4

INTERACTIVE PREFERENCE LEARNING: A PERSONALIZED NUTRIENT-BASED RECIPE RECOMMENDATION SYSTEM

4.1 Introduction

Previous two chapters developed user-centric recommenders using passively recorded offline user interaction data, such as implicit feedback (Chapter 2) and software usage history (Chapter 3). However, these data sources are hardly available in many domains. For example, users’ food consumption history necessary for food preference modeling is often hard to record [31]. In addition, offline data does not reflect aspirational preferences, e.g., people’s nutritional and health objectives. To address these limitations, this chapter explores the opportunity of actively interacting with users as means to learn their current and aspirational preferences. Specifically, we design and evaluate a recipe recommender, **Yum-me**, that satisfies users’ fine-grained food preferences and nutritional expectations without relying on consumption history. Our system addresses limitations of traditional approaches used to suggest food alternatives that cater to individuals’ health goals [189, 160, 120], including on-boarding surveys and food journaling:

- **Preferences elicited by surveys are coarse-grained.** A typical on-boarding survey asks a number of multi-choice questions about general food preferences. For example, PlateJoy [120], a daily meal planner app, elicits preferences for healthy goals and dietary restrictions with the following questions:

(1) *How do you prefer to eat? No restrictions, dairy free, gluten free, kid friendly, pescatarian, paleo, vegetarian...*

(2) *Are there any ingredients you prefer to avoid? avocado, eggplant, eggs, seafood, shellfish, lamb, peanuts, tofu....*

While the answers to these questions can and should be used to create a rough dietary plan and avoid clearly unacceptable choices, they do not generate recipe recommendations that cater to each person's fine-grained food preferences, and this may contribute to their lower than desired recommendation-acceptance rates (Section 4.6.3).

- **Food journaling approach suffers from cold-start problem and is hard to maintain.** For example, Nutrino [114], a personal meal recommender, asks users to log their daily food consumption and learn users' fine-grained food preferences. As is typical of systems relying on user-generated data, food journaling suffers from the *cold-start* problem, where recommendations cannot be made or are subject to low accuracy when the user has not yet generated a sufficient amount of data. For example, a previous study showed that an active food-journaling user makes about 3.5 entries per day [31]. It would take a non-trivial amount of time for the system to acquire sufficient data to make recommendations, and the collected samples may be subject to sampling biases as well [31, 86]. Moreover, the photo food journaling of all meals is a habit difficult to adopt and maintain, and therefore is not a generally applicable solution to generate complete food inventories [31].

We address these limitations by leveraging people's apparent desire to en-

gage with food photos¹ to create a more user-friendly medium for asking visually-based diet-related questions. The recommender learns users' fine-grained food preferences through a simple quiz-based visual interface [173] and then attempts to generate recipe recommendations that cater to the user's health goals, food restrictions, as well as personal appetite for food. It can be used by people who have food restrictions, such as vegetarian, vegan, kosher, and halal. Particularly, we focus on the health goals in the form of nutritional expectations, e.g., adjusting calories, protein, and fat intake. The mapping from health goals to nutritional expectations can be accomplished by professional nutritionists or personal coaches and is out of the scope of this chapter. We leave it as future work. For the visual interface [173], we propose a novel online learning framework to learn users' preferences for a large number of food items through a modest number of interactions. Our online learning approach balances exploitation-exploration and takes advantage of food image similarities. To the best of our knowledge, this is the first interface and algorithm that learns users' food preferences through real-time interactions without requiring food consumption history.

For such an online learning algorithm to work, one of the most critical components is a robust food image analysis model. Towards that end, as an additional contribution of this chapter, we present a novel and unified food image analysis model, named **FoodDist**. Based on deep convolutional networks and multi-task learning [91, 17], FoodDist is the best-of-its-kind Euclidean distance embedding for food images, in which similar food items have smaller distances while dissimilar food items have larger distances. FoodDist allows

¹Collecting, sharing and appreciating high quality, delicious-looking food images is a growing fashion in our everyday lives. For example, food photos are immensely popular on Instagram (*#food* has over 177M posts and *#foodporn* has over 91M posts at the time of writing).

the recommender to learn users’ fine-grained food preferences accurately via similarity assessments on food images. Besides preference learning, FoodDist can be applied to other food-image-related tasks, such as food image detection, classification, retrieval, and clustering. We benchmark FoodDist with Food-101 dataset [17], the largest dataset for food images. The results suggest the superior performance of FoodDist over prior approaches [173, 107, 17]. FoodDist is available at <https://github.com/ylongqi/fooddist>.

We evaluate our online learning framework in a field study with 227 anonymous users, and we show that it is able to predict the food items that a user likes or dislikes with high accuracy. Furthermore, we evaluate the desirability of Yum-me recommendations end-to-end through a 60-person user study, where each user rates the recipe recommendations made by Yum-me relative to those made using a traditional survey-based approach. The study results show that, compared to the traditional survey based recommender, our system significantly improves the acceptance rate of the recommended recipes by 42.63%. We see Yum-me as a complement to the existing food preference elicitation approaches. It further filters the food items selected by a traditional onboarding survey based on users’ fine-grained tastes for food and allows a system to serve tailored recommendations upon its first use. We discuss some potential use cases in Section 4.7. The implementation of Yum-me is available at <https://github.com/ylongqi/yumme>.

The rest of the chapter is organized as follows. After discussing related work in Section 4.2, we introduce the structure of Yum-me and our backend database in Section 4.3. In Section 4.4, we describe the algorithmic details of the proposed online learning algorithm, followed by the architecture of Food-

Dist model in Section 4.5. The evaluation results of each component, as well as the recommender are presented in Section 4.6. Finally, we discuss the limitations, potential impact and real world applications in Section 4.7 and conclude in Section 4.8.

4.2 Related work

This chapter benefits from, and is relevant to, multiple research threads: (1) healthy meal recommender system, (2) cold-start problem and preference elicitation, (3) pairwise algorithms for recommendation, and (4) food image analysis, which will be surveyed in detail next.

4.2.1 Healthy meal recommender system

Traditional food and recipe recommender systems learn users’ dietary preferences from their online activities, including ratings [51, 53, 64, 45], clicks [145, 56], and browsing history [157, 160, 114]. For example, Svensson et al. [145] built a social navigation system that recommends recipes based on users’ previous choices; Pinxteren et al. [160] proposed to learn a recipe similarity measure from crowd card-sorting and made recommendations based on the self-reported data; Harvey et al. [64] and Elswailer et al. [45] generated healthy meal plans based on users’ ratings towards a set of recipes and the nutritional requirements calculated for the persona. In addition, previous recommenders also seek to incorporate users’ food consumption history recorded by food journaling systems (e.g., taking food images [31] or writing down ingredients and

meta-information [160]).

The above systems, albeit are able to learn fine-grained food preferences, share a common limitation: their recommendations are not effective for a user until she generates enough data. Therefore, most commercial applications, such as Zipongo [190] and Shopwell [135], adopt onboarding surveys to more quickly elicit **coarse-grained** food preferences. For example, Zipongo’s questionnaires [190] ask users about their nutrient intake, lifestyle, habits, and food preferences, and then make day-to-day and week-to-week healthy meal recommendations. And ShopWell’s survey [135] is designed to avoid certain food allergens, e.g., gluten, fish, corn, or poultry, and find meals that match certain lifestyles, e.g., healthy pregnancy or athletic training.

Comparing to existing approaches, Yum-me enables a rapid elicitation of users’ fine-grained food preferences for immediate healthy meal recommendations. Based on an online learning framework [173], Yum-me infers users’ preferences for each single food item in a large food dataset, and leverage these learned preferences to recommend recipes catering to individual user’s nutritional aspirations.

4.2.2 Cold-start problem and preference elicitation

To alleviate the cold-start problem mentioned above, several models of preference elicitation have been proposed in recent years. The most prevalent method of elicitation is to train decision trees to poll users in a structured fashion [121, 57, 188, 35, 144]. These questions were selected either in advance and remain static [121] or dynamically based on real-time user feed-

back [57, 188, 35, 144]. Previous work also explored the possibility of eliciting item ratings directly from users [184, 26]. This process can either be carried at item- [184] or category- [26] level.

Existing preference elicitation methods mostly focus on the domain of movie recommendations [144, 121, 26, 184] and visual commerce [35] (e.g., cars and cameras), where items can be categorized based on readily available metadata. When it comes to real dishes, however, categorical data (e.g., cuisines) and other associated information (e.g., cooking time) possess a much weaker connection to a user’s food preferences. Therefore, in this chapter, we leverage the visual representation of each meal so as to better capture the process through which people make diet decisions.

4.2.3 Pairwise algorithms for recommendation

Pairwise approaches [123, 117, 124, 71, 174, 163, 165] are widely studied in recommender system literature. For example, Bayesian Personalized Ranking (BPR) [124, 123] and Weighted Approximate-Rank Pairwise (WARP) loss [163] are two representative and popular approaches under this category. Such algorithms have successfully powered many state-of-the-art systems [71, 165]. In terms of the cold-start scenario, Park et al. [117] developed a pairwise method to leverage users’ demographic information in recommending new items.

Compared to previous methods, our problem setting is fundamentally different in the sense that **Yum-me** elicits preferences in an active manner where the input is incremental and contingent on the previous decisions made by the algorithm, whereas prior work focuses on the static circumstances where the

training data is available up-front, and there is no need for the system to actively interact with the user.

4.2.4 Food image analysis

The task of analyzing food images is very important for many dietary applications that actively or passively collect food images from mobile [31] and wearable [7, 152, 111] devices. The estimation of food intake and its nutritional information provides detailed records of people’s dietary history [113]. Previous work mainly conducted the analysis by leveraging crowdsourcing [113, 156] or computer vision algorithms [17, 107].

Noronha et al. [113] crowdsourced nutritional analysis of food images by leveraging the wisdom of untrained crowds. The study demonstrated the possibility of estimating a meal’s calories, fat, carbohydrates, and protein by aggregating opinions from a large number of people. Turner-McGrievy et al. [156] instructed a crowd to rank the healthiness of several food items and validated the results against the ground truth provided by trained observers. Although this approach has been justified to be accurate, it inherently requires human resources that restrict it from scaling up to a large number of users.

To overcome the limitations of crowdsourcing and automate the analysis process, prior work built computer vision algorithms for food image classification [17, 107, 83, 12], retrieval [85], and nutrient estimation [107, 143, 23, 69]. However, most of the previous work [17] leveraged hand-crafted image features. And existing approaches were only evaluated in controlled environment, such as in a specific restaurant [12] or for a particular type of cuisine [83]. These



Figure 4.1: An overview of Yum-me. This figure shows three sample scenarios in which Yum-me can be used: desktop browser, mobile, and smart watch. The fine-grained dietary profile is used to re-rank and personalize recipe recommendations.

models’ performance might degrade when applied to food images in the wild.

In this chapter, we design FoodDist using deep convolutional neural network based multitask learning [22], which has been shown to be successful in improving the generalization power and performance in several applications [186, 33]. The main challenge of multitask learning is to design appropriate network structures and sharing mechanisms across tasks. With our proposed network structure, we show that, compared to prior approaches, FoodDist achieves superior performance when applied to the largest available real-world food image dataset [17].

4.3 Yum-me system design

Our personalized nutrient-based recipe recommendation system, Yum-me, operates over a given inventory of food items and suggests the items that will appeal to a user’s palate and meet her nutritional expectations and dietary restric-

tions. A high-level overview of Yum-me’s recommendation process is shown in Fig. 4.1 and briefly described as follows:

- *Step 1:* A user answers a simple survey to specify her dietary restrictions and nutritional expectations. This information is used by Yum-me to filter food items and create an initial set of recommendation candidates.
- *Step 2:* The user then uses an adaptive visual interface to express her fine-grained food preferences through simple comparisons of food items. The learned preferences are used to further re-rank the recommendations presented to her.

In the rest of this section, we describe our backend large-scale food database and aforementioned two recommendation steps: (1) a user survey that elicits dietary restrictions and nutritional expectations, and (2) an adaptive visual interface that elicits fine-grained food preferences.

4.3.1 Large scale food database

To account for the dietary restrictions in many cultures and religions, or people’s personal choices, we prepare a separate food database for each of the following dietary restrictions:

No restrictions, Vegetarian, Vegan, Kosher, Halal ²

²Our system is not restricted to these five dietary restrictions and we will extend the system functionalities to other categories in the future.

Table 4.1: The size of databases for different diet types. Unit: number of unique recipes.

Database	Original size	Final size
No restriction	9405	7938
Vegetarian	10000	6713
Vegan	9638	6013
Kosher	10000	4825
Halal	10000	5002

For each diet type, we scraped 10,000 main dish recipes along with their images and metadata (e.g., ingredients, nutrients, tastes, etc.) from the Yummly API [182]. The total number of recipes is around 50,000. In order to customize food recommendations for people with specific dietary restrictions, e.g., vegetarian and vegan, we filter recipes by setting the *allowedDiet* parameter in the search API. For kosher or halal, we explicitly rule out certain ingredients by setting *excludedIngredient* parameter. The excluded ingredients include:

- **Kosher:** pork, rabbit, horse meat, bear, shellfish, shark, eel, octopus, octopuses, moreton bay bugs, and frog.
- **Halal:** pork, blood sausage, blood, blood pudding, alcohol, grain alcohol, pure grain alcohol, and ethyl alcohol.

One challenge in using such a public food image API is that many recipes returned by the API contain non-food images and incomplete nutritional information. Therefore, we further filter the items with the following criteria: the recipe should have nutritional information of calories, protein and fat, and at least one food image. In order to automate this process, we build a binary classifier based

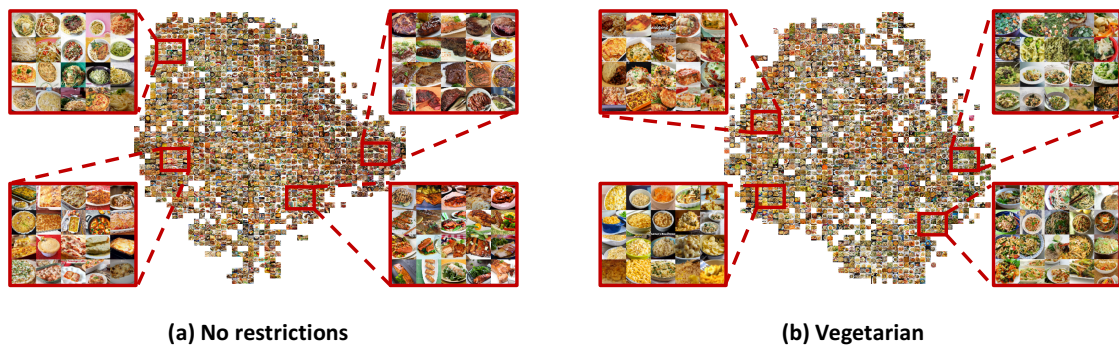


Figure 4.2: An overview of two sample databases: (a) for users without dietary restrictions and (b) for vegetarian users.

on a deep convolutional neural network to filter out non-food images. As suggested by [107], we treat the whole training set of Food-101 dataset [17] as one generic *food* category and sampled the same number of images (75,750) from the ImageNet dataset [37] as our *non-food* category. We took the pretrained VGG CNN model [136] and replaced the final 1000 dimensional softmax with a single logistic node. For the validation, we used the Food-101 testing dataset along with the same number of images sampled from ImageNet (25,250). We trained the binary classifier using the Caffe framework [79] and it reached 98.7% validation accuracy. We applied the criteria to all the datasets and the final statistics are shown in Table. 4.1.

Fig. 4.2 shows the visualizations of the collected datasets. For each of the recipe images, we embed it into an 1000-dimensional feature space using Food-Dist (described later in Section 4.5) and then project all the images onto a 2-D plane using t-Distributed Stochastic Neighbor Embedding(t-SNE) [159]. For visibility, we further divide the 2-D plane into several blocks; from each of which, we sample a representative food image residing in that block to present in the figure. Fig. 4.2 demonstrates the large diversity and coverage of the collected

datasets. Also, the embedding results clearly demonstrate the effectiveness of FoodDist in grouping similar food items together while pushing dissimilar items away. This is important to the performance of Yum-me (Section 4.6.3).

4.3.2 User survey

The user survey is designed to elicit a user’s high-level dietary restrictions and nutritional expectations. A user can specify her dietary restrictions among the five categories mentioned above and indicate her nutritional expectations in terms of the desired amount of **calories**, **protein** and **fat**. We choose these nutrients for their high relevance to many common health goals, such as weight control [46], sports performance [21], etc. We provide three options for each of these nutrients: **reduce**, **maintain**, and **increase**. The user’s diet type is used to select an appropriate food dataset, and the food items in the dataset are further ranked by their suitability to the user’s nutritional goals.

To measure the suitability of food items given nutritional expectations, we rank the recipes in terms of different nutrients in both *ascending* and *descending* order, such that each recipe is associated with six ranking values, i.e., $r_{\text{calories},a}$, $r_{\text{calories},d}$, $r_{\text{protein},a}$, $r_{\text{protein},d}$, $r_{\text{fat},a}$ and $r_{\text{fat},d}$, where a and d stand for *ascending* and *descending* respectively. The final suitability value for each recipe given the health goal is calculated as follows:

$$u = \sum_{n \in \mathbb{U}} \alpha_{n,a} r_{n,a} + \sum_{n \in \mathbb{U}} \alpha_{n,d} r_{n,d}, \quad (4.1)$$

where $\mathbb{U} = \{\text{calories}, \text{protein}, \text{fat}\}$. The indicator coefficient $\alpha_{n,a} = 1 \iff$ nutrient n is rated as *reduce* and $\alpha_{n,d} = 1 \iff$ nutrient n is rated as *increase*. Otherwise $\alpha_{n,a} = 0$ and $\alpha_{n,d} = 0$. If a user’s goal is to maintain all nutrients, then all recipes

are given equal rankings. Eventually, given a user’s responses to the survey, we rank the suitability of all the recipes in the corresponding database and select top- M items (around top 10%) as the candidate pool of proper recipes for this user. In our initial prototype, we set $M = 500$.

4.3.3 Adaptive visual interface

Based on the food suitability ranking, a candidate pool of proper recipes is created. However, not all the recipes in this candidate pool may suit the user’s palate. Therefore, we design an adaptive visual interface to further identify recipes that cater to the user’s taste through eliciting their fine-grained food preferences. We propose to learn fine-grained food preferences by presenting food images to the user and asking her to choose the ones that look delicious.

Formally, the food preference learning task can be defined as follows. Given a large **target set** of food items \mathbb{S} , we represent a user’s preferences as a distribution over all the possible food items, i.e., $\mathbf{p} = [p_1, \dots, p_{|\mathbb{S}|}]$, $\sum_i p_i = 1$, where each element p_i denotes the user’s favorable scale for item i . Since the number of items, $|\mathbb{S}|$, is usually quite large and intractable to elicit individually from the user³, the approach we take is to adaptively choose a specific and much smaller **subset** \mathbb{V} to present to the user, and propagate the user’s preferences for those items to the rest items based on visual similarity. Specifically, as Fig. 4.1 shows, the preference elicitation process can be divided into two phases:

Phase I: In each of the first 2 iterations, we present ten food images and ask the user to tap on all the items that look delicious to them.

³The target set is often the whole food database that different applications use. For example, the size of Yummly database can be up to 1-million [182].

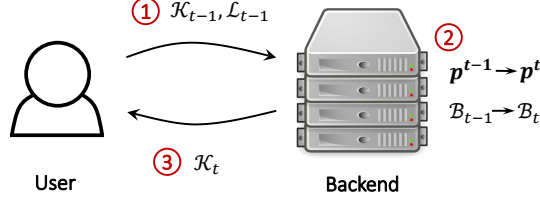


Figure 4.3: User-system interaction at iteration t .

Phase II: In each of the subsequent iterations, we present a pair of food images and ask the user to either compare the food pair and tap on the one that looks delicious to her or tap on “Yuck” if neither of the items appeal to her taste.

In order to support the preference elicitation process, we design a novel exploration-exploitation online learning algorithm (Section 4.4) built on a state-of-the-art food image embedding model (Section 4.5).

4.4 Online learning framework

We model the interactions between the user and our backend system at iteration t , ($t \in \mathcal{R}^+$, $t = 1, 2, \dots, T$) as Fig. 4.3 shows. The symbols used in our algorithm are defined as follows:

- \mathcal{K}_t : The set of food items presented to the user at iteration t ($\mathcal{K}_0 = \emptyset$).
 $\forall k \in \mathcal{K}_t, k \in \mathbb{S}$;
- \mathcal{L}_{t-1} : The set of food items that the user prefers (selects) among $\{k | k \in \mathcal{K}_{t-1}\}$.
 $\mathcal{L}_{t-1} \subseteq \mathcal{K}_{t-1}$;
- $\mathbf{p}^t = [p_1^t, \dots, p_{|\mathbb{S}|}^t]$: The user’s preference distribution over all food items at iteration t , where $\|\mathbf{p}^t\|_1 = 1$. \mathbf{p}^0 is initialized as $p_i^0 = \frac{1}{|\mathbb{S}|}$;

- \mathcal{B}_t : The set of food images that have already been explored until iteration t ($\mathcal{B}_0 = \emptyset$). $\mathcal{B}_i \subseteq \mathcal{B}_j (i < j)$;
- $\mathcal{F} = \{f(x_1), \dots, f(x_{|\mathbb{S}|})\}$: The set of feature vectors of food images $x_i (i = 1, \dots, |\mathbb{S}|)$ extracted by a feature extractor, denoted by f . We use FoodDist (Section 4.5) as the feature extractor.

Based on the workflow depicted in Fig. 4.3, for each iteration t , the backend system updates vector \mathbf{p}^{t-1} to \mathbf{p}^t and set \mathcal{B}_{t-1} to \mathcal{B}_t based on users' selections \mathcal{L}_{t-1} and previous image set \mathcal{K}_{t-1} . After that, it decides the set of images to be presented to the user (i.e., \mathcal{K}_t). Our food preference elicitation framework can be formalized in Algorithm. 1. The core procedures are *update* and *select*, which are described in the following subsections for more details.

Algorithm 1: Food Preference Elicitation Framework

Data: $\mathbb{S}, \mathcal{F} = \{f(x_1), \dots, f(x_{|\mathbb{S}|})\}$

Result: \mathbf{p}^T

```

1  $\mathcal{B}_0 = \emptyset, \mathcal{K}_0 = \emptyset, \mathcal{L}_0 = \emptyset, \mathbf{p}^0 = [\frac{1}{|\mathbb{S}|}, \dots, \frac{1}{|\mathbb{S}|}]$  ;
2 for  $t \leftarrow 1$  to  $T$  do
3    $[\mathcal{B}_t, \mathbf{p}^t] \leftarrow \text{update}(\mathcal{K}_{t-1}, \mathcal{L}_{t-1}, \mathcal{B}_{t-1}, \mathbf{p}^{t-1})$  ;
4    $\mathcal{K}_t \leftarrow \text{select}(t, \mathcal{B}_t, \mathbf{p}_t)$  ;
5   if  $t$  equals  $T$  then
6     return  $\mathbf{p}^T$ 
7   else
8      $\text{ShowToUser}(\mathcal{K}_t)$  ;
9      $\mathcal{L}_t \leftarrow \text{WaitForSelection}()$  ;
```

4.4.1 User state update

Based on the user's selections \mathcal{L}_{t-1} and the image set \mathcal{K}_{t-1} , the *update* module renews the user's state from $\{\mathcal{B}_{t-1}, \mathbf{p}^{t-1}\}$ to $\{\mathcal{B}_t, \mathbf{p}^t\}$. Our intuition and assumption

behind following algorithm design is that people tend to have close preferences for similar food items.

Updating the preference vector \mathbf{p}^t

Our strategy of updating the preference vector \mathbf{p}^t is inspired by the Exponentiated Gradient Algorithm in bandit settings (EXP3) [9]. Specifically, at iteration t , each p_i^t in the vector \mathbf{p}^t is updated by:

$$p_i^t \leftarrow p_i^{t-1} \times e^{\frac{\beta u_i^{t-1}}{p_i^{t-1}}}, \quad (4.2)$$

where β is the exponentiated coefficient that controls update speed and $\mathbf{u}^{t-1} = \{u_1^{t-1}, \dots, u_{|\mathbb{S}|}^{t-1}\}$ is the update vector used to adjust each preference value.

In order to calculate update vector \mathbf{u} , we formalize the user's selection process as a data labeling problem [187] where for item $i \in \mathcal{L}_{t-1}$, label $y_i^{t-1} = 1$ and for item $j \in \mathcal{K}_{t-1} \setminus \mathcal{L}_{t-1}$, label $y_j^{t-1} = -1$. Thus, the label vector $\mathbf{y}^{t-1} = \{y_1^{t-1}, \dots, y_{|\mathbb{S}|}^{t-1}\}$ provided by the user is:

$$y_i^{t-1} = \begin{cases} 1 & : i \in \mathcal{L}_{t-1} \\ 0 & : i \notin \mathcal{K}_{t-1} \\ -1 & : i \in \mathcal{K}_{t-1} \setminus \mathcal{L}_{t-1} \end{cases} \quad (4.3)$$

For the update vector \mathbf{u} , we expect that it is close to the label vector \mathbf{y} but with smooth propagation of label values to nearby neighbors (For convenience, we omit superscript that denotes the current iteration). The update vector \mathbf{u} can be regarded as a soften label vector as compared to \mathbf{y} . To make the solution more computationally tractable, for each item i with $y_i \neq 0$, we construct a locally connected undirected graph G^i as Fig. 4.4 shows: $\forall j \in \mathbb{S}$, add an edge (i, j) if

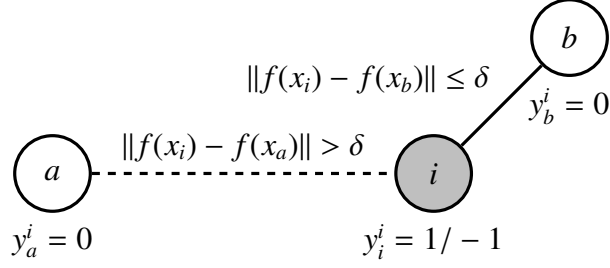


Figure 4.4: The locally connected graph with item i .

$\|f(x_i) - f(x_j)\| \leq \delta$. The labels \mathbf{y}^i for vertices s_j in graph G^i are calculated as $y_j^i = 0 (j = 1, \dots, |\mathbb{S}| \setminus i), y_i^i = y_i$.

For each locally connected graph G^i , we fix u_i^i value as $u_i^i = y_i^i$ and propose the following regularized optimization method to compute other elements ($\forall u_j^i, j \neq i$) of the update vector \mathbf{u}^i , which is inspired by the traditional label propagation method [187].

Consider the problem of minimizing the following objective function $Q(\mathbf{u}^i)$:

$$\min_{\mathbf{u}^i} Q(\mathbf{u}^i) = \sum_{j=1, j \neq i}^{|\mathbb{S}|} w_{ij} (y_j^i - u_j^i)^2 + \sum_{j=1, j \neq i}^{|\mathbb{S}|} (1 - w_{ij}) (u_j^i - y_j^i)^2, \quad (4.4)$$

where w_{ij} represents the similarity measure between food item s_i and s_j , as eqn. 4.5 shows.

$$w_{ij} = \begin{cases} e^{-\frac{1}{2\alpha^2} \|f(x_i) - f(x_j)\|^2} & : \|f(x_i) - f(x_j)\| \leq \delta \\ 0 & : \|f(x_i) - f(x_j)\| > \delta \end{cases} \quad (4.5)$$

where $\alpha^2 = \frac{1}{|\mathbb{S}|^2} \sum_{i,j \in \mathbb{S}} \|f(x_i) - f(x_j)\|^2$

The first term of the objective function $Q(\mathbf{u}^i)$ is a *smoothness constraint* as the update value for similar food items should not change too much. The second term is a *fitting constraint*, which makes \mathbf{u}^i close to the initial labeling assigned by the user (i.e., \mathbf{y}^i). However, unlike [187], in our algorithm, the trade-off between

these two constraints is dynamically adjusted by the similarity between item i and j where similar pairs are weighed more with smoothness and dissimilar pairs are forced to be close to the initial labeling.

We calculate the optimal u_j^i by taking the partial derivative of $Q(\mathbf{u}^i)$ with respect to different u_j^i :

$$\frac{\partial Q(\mathbf{u}^i)}{u_{j,j \neq i}^i} = 2w_{ij}(u_j^i - u_i^i) + 2(1 - w_{ij})(u_j^i - y_j^i) = 0 \quad (4.6)$$

And as $u_i^i = y_i^i$,

$$u_j^i = w_{ij}u_i^i = w_{ij}y_i^i (j = 1, 2, \dots, |\mathbb{S}|) \quad (4.7)$$

Eventually, the original update vector \mathbf{u} is calculated as $\mathbf{u} = \sum_i \mathbf{u}^i$.

Updating the explored food image set \mathcal{B}_t

In order to balance the *exploitation* and *exploration* in the images selection phase, we maintain a set \mathcal{B}_t that keeps track of all similar food items that have already been visited by user and the updating rule for \mathcal{B}_t is as follows:

$$\mathcal{B}_t \leftarrow \mathcal{B}_{t-1} \cup \{i \in \mathbb{S} | \min_{j \in \mathcal{K}_{t-1}} \|f(x_i) - f(x_j)\| \leq \delta\} \quad (4.8)$$

The pseudo code for the *update* module is shown in Algorithm.2.

Algorithm 2: User state update Algorithm

```
1 Function update ( $\mathcal{K}_{t-1}, \mathcal{L}_{t-1}, \mathcal{B}_{t-1}, \mathbf{p}^{t-1}$ )  
   input :  $\mathcal{K}_{t-1}, \mathcal{L}_{t-1}, \mathcal{B}_{t-1}, \mathbf{p}^{t-1}$   
   output:  $\mathcal{B}_t, \mathbf{p}^t$   
  
2    $\mathbf{u} = [0, \dots, 0], \mathcal{B}_t = \mathcal{B}_{t-1}, \mathbf{p}^t = \mathbf{p}^{t-1}$   
3   for  $i \leftarrow 1$  to  $|\mathbb{S}|$  do  
4     // preference update  
5     for  $s_j$  in  $\mathcal{K}_{t-1}$  do  
6        $u_i \leftarrow u_i + (-1)^{\mathbb{1}(j \in \mathcal{L}_{t-1})-1} w_{ij}$   
7        $p_i^t = p_i^{t-1} e^{\frac{\beta u_i}{p_i^{t-1}}}$   
8       // explored image set update  
9       if  $\min(\|f(x_i) - f(x_j)\|, \forall j \in \mathcal{K}_{t-1}) \leq \delta$  then  
10         $\mathcal{B}_t \leftarrow \mathcal{B}_t \cup \{i\}$   
11   // normalize  $\mathbf{p}^t$  s.t.  $\|\mathbf{p}^t\|_1 = 1$   
12   normalize ( $\mathbf{p}^t$ )
```

4.4.2 Images selection

After updating the user state, the *select* module then picks food images to be presented in the next round. The selection process trade-offs between exploration and exploitation.

Food Exploration

For each of the first two iterations, we select ten different food images through *K-means++* [8] algorithm, a seeding method used in *K-means clustering*. It guarantees that the selected items are evenly distributed in the feature space. For our use case, *K-means++* algorithm is summarized in Algorithm.3.

Algorithm 3: Kmeans++ Algorithm for Exploration

```
1 Function k-means-pp ( $\mathbb{S}, n$ )  
   input :  $\mathbb{S}, n$   
   output:  $\mathcal{K}_t$   
2    $\mathcal{K}_t = \text{random}(\mathbb{S})$   
3   while  $|\mathcal{K}_t| < n$  do  
4      $\text{prob} \leftarrow [0, \dots, 0]_{|\mathbb{S}|}$   
5     for  $i \leftarrow 1$  to  $|\mathbb{S}|$  do  
6        $\text{prob}_i \leftarrow \min(\|f(x_i) - f(x_j)\|^2 | \forall j \in \mathcal{K}_t)$   
7        $\text{sample } m \in \mathbb{S} \text{ with probability } \propto \text{prob}_m$   
8        $\mathcal{K}_t \leftarrow \mathcal{K}_t \cup \{m\}$ 
```

Food Exploitation-Exploration

Starting from the third iteration, the user is asked to make pairwise comparisons between food images. To balance exploitation and exploration, we select one image from the area with higher preference value based on the current \mathbf{p}' and the other one from the *unexplored* area, i.e., $\mathbb{S} \setminus \mathcal{B}_t$. (Both selections are **random** given a subset of food items). The detailed images selection algorithm is summarized in Algorithm 4.

Algorithm 4: Images Selection Algorithm - select

```
1 Function select ( $t, \mathcal{B}_t, \mathbf{p}'$ )  
   input :  $t, \mathcal{B}_t, \mathbf{p}'$   
   output:  $\mathcal{K}_t$   
2    $\mathcal{K}_t = \emptyset$   
3   if  $t \leq 2$  then  
4      $\mathcal{K}_t \leftarrow \text{k-means-pp}(\mathbb{S}, 10)$  // K-means++  
5   else  
6     // 99th percentile (top 1%)  
7      $\text{threshold} \leftarrow \text{percentile}(\mathbf{p}', 99)$   
8      $\text{topSet} \leftarrow \{s_i \in \mathbb{S} | p'_i \geq \text{threshold}\}$   
9      $\mathcal{K}_t \leftarrow [\text{random}(\text{topSet}), \text{random}(\mathbb{S} \setminus \mathcal{B}_t)]$ 
```

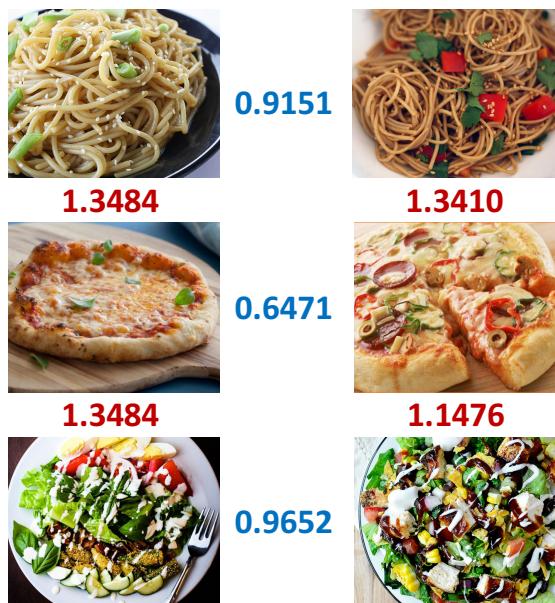


Figure 4.5: An Euclidean distance embedding of FoodDist. This figure shows the pairwise euclidean distances between food image representations. A distance of 0.0 means that two food items are identical and a distance of 2.0 represents that the image contents are completely different. In this example, if the threshold is set to 1.0, then all the images can be correctly classified.

4.5 FoodDist: food image embedding

The goal of FoodDist is to learn a **feature extractor (embedding)** f that projects images to an N dimensional embedding space where Euclidean distances between feature vectors reflect the similarities between food items, as Fig. 4.5 shows. Formally speaking, if image x_1 is more similar to image x_2 than image x_3 , then $\|f(x_1) - f(x_2)\| < \|f(x_1) - f(x_3)\|$.

We build FoodDist based on recent advances in deep Convolutional Neural Networks (CNN), which provides a powerful framework for automatic feature learning. Traditional feature representations for images are mostly hand-crafted. For example, the SIFT (Scale Invariant Feature Transform) [101] fea-

ture descriptor is invariant to changes in object scale and illumination, thereby improving the generalizability of the trained model. However, in the face of highly diverse image characteristics, the one-size-fits-all feature extractor performs poorly. In contrast, deep learning adapts to different image characteristics and extracts features that are most discriminative for a task [122].

A feature extractor for food images can be learned through classification and metric learning, or through multitask learning, which concurrently performs these two tasks. We demonstrate that our proposed multitask learning approach enjoys the benefits of both classification and metric learning, and achieves the best performance.

4.5.1 Learning with classification

One common way to learn a feature extractor for labeled data is to train a neural network to perform classification (i.e., mapping input to labels), and takes the output of a hidden layer as the feature representations. Specifically, we use a feedforward deep CNN with n -layers (as the upper half of the Fig. 4.6 shows):

$$F(x) = g_n(g_{n-1}(\dots g_i(\dots g_1(x) \dots))), \quad (4.9)$$

where $g_i(\cdot)$ represents the computation of the i -th layer (e.g., convolution, pooling, fully-connected, etc.), and $F(x)$ is the output class label. The difference between the output class label and the ground truth (i.e., the error) is back-propagated throughout the whole network from layer n to the layer 1. We can take the output of the layer $n - 1$ as the feature representation of x , which is equivalent to having a feature extractor f as:

$$f(x) = g_{n-1}(\dots g_i(\dots g_1(x)\dots)) \quad (4.10)$$

Usually, the last few layers are fully-connected layers, and the last layer $g_n(\cdot)$ is roughly equivalent to a linear classifier that is built on the features $f(x)$ [76]. Therefore, $f(x)$ is discriminative in separating instances under different categorical labels, and the Euclidean distances between normalized feature vectors can reflect the similarities between images.

4.5.2 Metric learning

Different from the classification approach, where the feature extractor is a by-product, metric learning proposes to learn the distance embedding directly from the paired inputs of similar and dissimilar examples. Prior work [173] used a *Siamese network* to learn a feature extractor for food images. The structure of a Siamese network resembles that in Fig. 4.6 but without *Class label*, *Fully connected*, *101* and *Softmax Loss* layers. The inputs to the Siamese network are pairs of food images x_1, x_2 . The images pass through CNNs with shared weights and the output of each network is regarded as the feature representation, i.e., $f(x_1)$ and $f(x_2)$, respectively. Our goal is for $f(x_1)$ and $f(x_2)$ to have a small distance value (close to 0) if x_1 and x_2 are similar food items; otherwise, they should have a larger distance value. The value of the contrastive loss is then back-propagated to optimize the Siamese network:

$$\mathcal{L}(x_1, x_2, l) = \frac{1}{2}lD^2 + \frac{1}{2}(1-l)\max(0, m-D)^2, \quad (4.11)$$

where similarity label $l \in \{0, 1\}$ indicates whether the input pair of food items x_1, x_2 are similar or not ($l = 1$ for similar, $l = 0$ for dissimilar), $m > 0$ is the margin

4.5.3 Multitask learning

Either of the learning methods above has its pros and cons. Learning with classification leverages the label information, but the network is not directly optimized towards our goal. As a result, although the feature vectors are learned to be separable in the linear space, the intra- and inter- categorical distances might still be unbalanced. On the other hand, metric learning is explicitly optimized for our final objective by pushing the distances between dissimilar food items apart beyond a margin m . Nevertheless, sampling the similar or dissimilar pairs loses valuable label information. For example, given a pair of items with different labels, we only consider the dissimilarity between the two categories they belong to, but overlook the fact that each item is also different from the remaining $n - 2$ categories, where n is the total number of categories.

In order to leverage the advantages of both methods, we propose a multitask learning design [76] for FoodDist. The idea of multitask learning is to share part of the model across tasks so as to improve the generalization ability of the learned model [76]. In our case, as Fig. 4.6 shows, we share the parameters between the classification network and Siamese network, and optimize them simultaneously. We use the base structure of the Siamese network and share the upper CNN with a classification network where the output of the CNN is fed into a cascade of a fully connected layer and a softmax loss layer. The final loss of the whole network is the weighted sum of the softmax loss $\mathcal{L}_{\text{softmax}}$ and contrastive loss $\mathcal{L}_{\text{contrastive}}$:

$$\mathcal{L} = \omega \mathcal{L}_{\text{softmax}} + (1 - \omega) \mathcal{L}_{\text{contrastive}} \quad (4.12)$$

Our benchmark results (Section 4.6.2) suggest that the feature extractor built with multitask learning: it achieves the best performance for both classification and Euclidean distance-based retrieval tasks.

4.6 Evaluation

We conduct user testing for the online learning framework and end-to-end recommender system (Yum-me), as well as offline evaluation for food image embedding model (FoodDist). Our hypothesis are summarized below:

- **H1:** Our online learning framework learns more accurate food preferences than baseline approaches.
- **H2:** FoodDist generates better similarity measure for food images than state-of-the-art embedding models.
- **H3:** Yum-me makes more accurate nutritionally appropriate recipe recommendations than traditional survey as it integrates coarse-grained item filtering with adaptively learned fine-grained food preferences.

In this section, we present (1) a user testing for the online learning framework (Section 4.6.1), (2) a benchmarking for FoodDist model using a large-scale real-world food image dataset (Section 4.6.2), and (3) an end-to-end lab user testing (Section 4.6.3).

4.6.1 User testing for the online learning framework

In order to evaluate the accuracy of our online learning framework, we conducted a field study with 227 anonymous users recruited from social networks and university mailing lists. The experiment was approved by the Institutional Review Board (ID: 1411005129) at Cornell University. All participants were required to use this system independently for three times. Each time the study consists of following two phases:

- *Training Phase.* A participant conducted the first T iterations of food image comparisons, and the system learnt and elicited preference vector \mathbf{p}^T based on the algorithms proposed in this chapter or baseline methods. We randomly picked T from the set $\{5, 10, 15\}$ at the beginning but made sure that each user experienced different values of T only once.
- *Testing Phase.* After T iterations of training, the participant entered the testing phase, which consists of 10 rounds of pairwise comparisons. We picked testing images based on the preference vector \mathbf{p}^T that learnt from online interactions: one of them was selected from the area that the user liked (i.e., the items with top 1% preference value) and the other one from the area that the user disliked (i.e., the items with bottom 1% preference value). Both images were picked at random from the unexplored items.

Prediction accuracy

In order to evaluate the effectiveness of the *user state update* and the *images selection* methods respectively. The experiment was 2×2 designed. For the *user state update* method, we compare the proposed *Label propagation*, *Exponentiated*

Gradient (LE) algorithm to the *Online Perceptron (OP)*, and for the *images selection* method, we compare the proposed *Exploration-Exploitation (EE)* algorithm to the *Random Selection (RS)*. Specifically, four frameworks were evaluated:

LE+EE: The online learning algorithm proposed in this chapter. It combines the ideas of Label propagation, Exponentiated Gradient algorithm for user state update and Exploitation-Exploration strategy for images selection.

LE+RS: The baseline algorithm that retains our method for user state update (**LE**) but Random Select images to present to the user without any exploitation or exploration.

OP+EE: As each item is represented by *1000 dim* feature vector, we can use regression to tackle this online learning problem (i.e., learning a weight vector \mathbf{w} such that $\mathbf{w}f(x_i)$ is higher for item i that the user prefers). Hence, we compare our method with an **Online Perceptron** algorithm that updates \mathbf{w} whenever it makes error, i.e., if $y_i\mathbf{w}f(x_i) \leq 0$, assign $\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{w}f(x_i)$, where y_i is the label for item i (pairwise comparison is regarded as binary classification such that the food item that the user selects is labeled as +1, and otherwise -1). In this algorithm, we retain our strategy of images selection (i.e., **EE**).

OP+RS: The baseline algorithm based on **OP+EE** but with **Random** images Selection strategy.

The participants were assigned to different algorithms completely at random. Among 227 participants in our study, 58 used algorithm **LE+EE**, and 57 used **OP+RS**. For the rest of users (112), half of them (56) tested **OP+EE** and the other half (56) tested **LE+RS**.

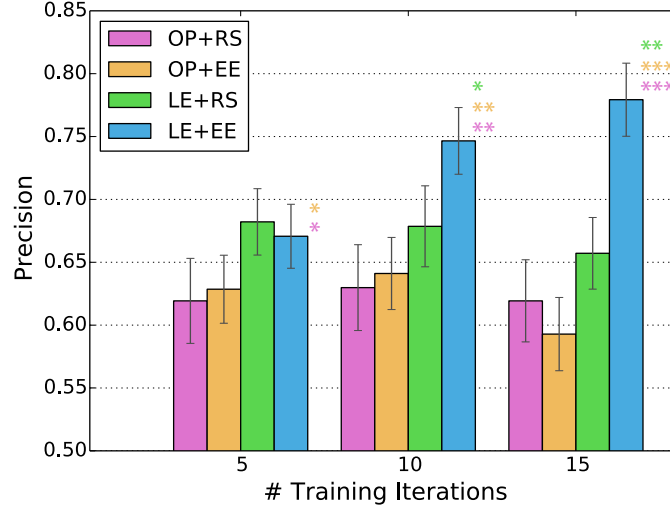


Figure 4.7: The prediction accuracy of different algorithms in various training settings (asterisks represent different levels of statistical significance: *** : $p < 0.001$, ** : $p < 0.01$, * : $p < 0.05$).

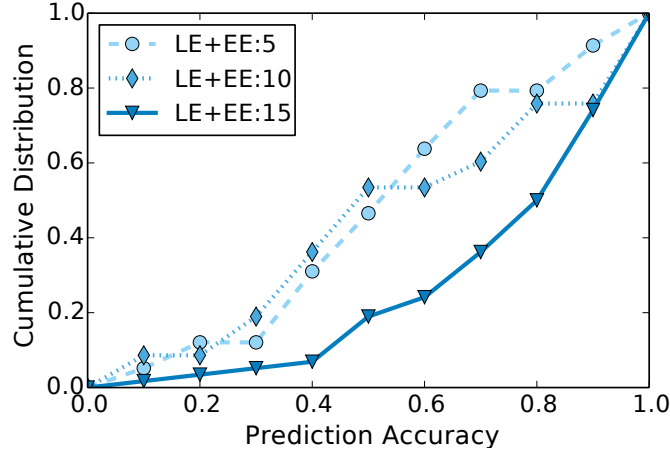


Figure 4.8: The cumulative distribution of the prediction accuracy of *LE+EE* algorithm (Numbers in the legend represent the values of T).

We calculate the prediction accuracy of each **individual user** and aggregate them based on the context that they encountered (i.e., the number of training iterations T and the algorithm settings). The prediction accuracies and their cumulative distributions are shown in Fig. 4.7, 4.8 and 4.9 respectively. These results demonstrate that our algorithm significantly outperforms baseline meth-

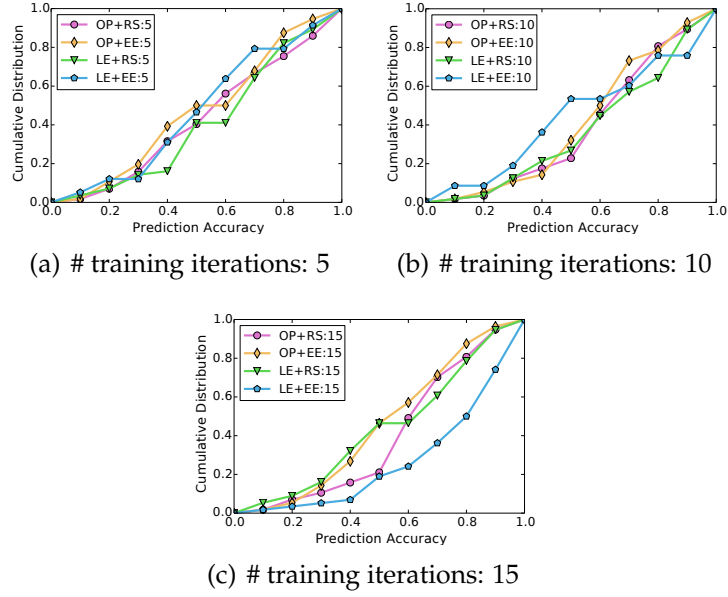


Figure 4.9: Comparison of the cumulative distribution of prediction accuracy across different algorithms.

ods. And it is more accurate with larger number of training iterations. In comparison, the prediction accuracy of the baseline methods decreases as users provide more information (larger T).

The main reasons why the baseline methods manifest suboptimal performance are: (1) Within a limited number of interactions, random selection can not maintain the knowledge that it has already learned from the user (exploitation), nor effectively explore the unknown areas (exploration). In addition, it's more likely for the baseline method to choose food items that are too similar for the user to effectively compare. (2) Online Perceptron (OP) tends to be underfitted. In our application, each food item is represented by *1000 dim* feature vector, and **OP** tries to learn a linear hyperplane based on a small number of training data points. And linearity can be an overly simplified assumption for features from a deep neural network.

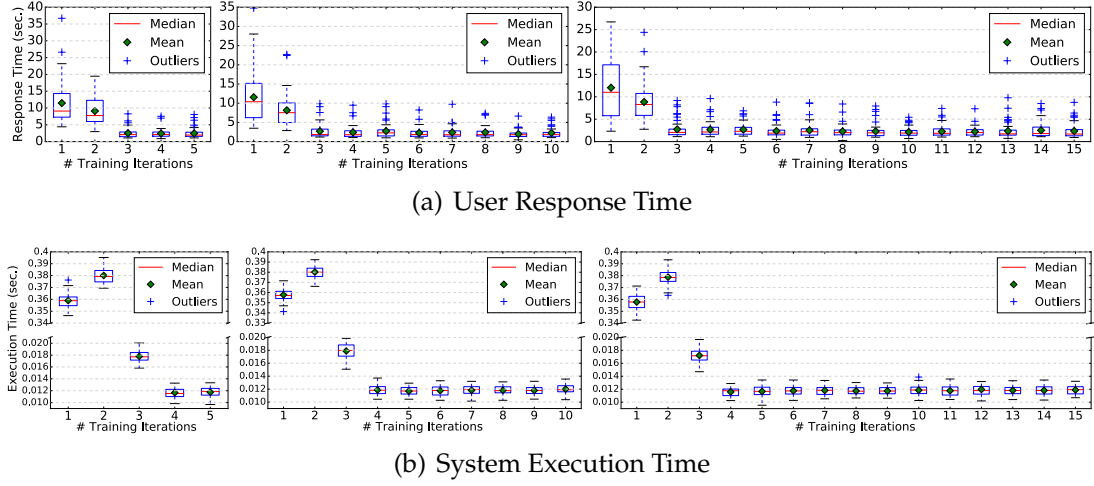


Figure 4.10: User response time and system execution time.

Table 4.2: Average duration to complete the training phase.

# Iter: 5	# Iter: 10	# Iter: 15
28.75s	39.74s	53.22s

System efficiency

Computing efficiency and user experiences are also important metrics in evaluating online learning systems. Therefore, we recorded the program execution time and user response time. As shown in Fig. 4.10(b), the program execution time is about 0.35s for the first two iterations, and less than 0.025s for the iterations afterwards⁴. Also, according to Fig. 4.10(a), the majority of users can make their decisions in less than 15s when comparing ten food images, whereas the payload for the pairwise comparison is less than 2 – 3s. In terms of the end-to-end system overhead (Table. 4.2), users can typically complete 15 training iterations within 53 seconds, which justify that our online learning framework is light-weight and user-friendly in efficiently eliciting food preferences.

⁴Our web system implementation is based on Amazon EC2 t2-micro Linux 64-bit instance

Qualitative feedback

After the study, some participants sent us emails regarding their experiences of using the adaptive visual interface. Most of the comments reflect the participants' satisfactions and that our system is able to engage the users throughout the elicitation process. For example, *"Now I'm really hungry and want a grilled cheese sandwich!"*, *"That was fun seeing tasty food at top of the morning."* and *"Pretty cool tool."*. However, they also highlight some limitations of our current prototype. For example, *"I am addicted to spicy food and it totally missed it. There may just not be enough spicy alternatives in the different dishes to pick up on it."* points out that the prototype is limited in the coverage of the food database.

4.6.2 Offline benchmarking for FoodDist

We developed FoodDist and baseline models (Section 4.5) using Food-101 training dataset, which contains 75,750 food images from 101 food categories (750 instances for each category) [17]. To the best of our knowledge, Food-101 is the largest and most challenging publicly available dataset for food images. We implemented models using Caffe [79] and experimented with two CNN architectures in our framework: AlexNet [91], which won the first place at ILSVRC2012 challenge, and VGG [136], which is the state-of-the-art CNN model. The inputs to the networks are image crops of size 224×224 (VGG) or 227×227 (AlexNet). They are randomly sampled from a pixelwise mean-subtracted image or its horizontal flip. In our benchmarking, we trained four different feature extractors: AlexNet+Learning with classification (**AlexNet+CL**), AlexNet+Multitask learning (**AlexNet+MT**), VGG+Learning with classification (**VGG+CL**) and

VGG+Multitask learning (**VGG+ML, FoodDist**). For the multitask learning framework, we sampled similar and dissimilar image pairs with 1:10 ratio from the Food-101 dataset based on the categorical labels [173]. The models were fine-tuned based on the networks pre-trained using the ImageNet dataset [37]. We used Stochastic Gradient Decent with a mini-batch size of 64, and each network was trained for 10^5 iterations. The initial learning rate is set to 0.001 and we used a weight decay of 0.0005 and momentum of 0.9.

We compared the performance of four feature extractors, including Food-Dist, with the state-of-the-art food image analysis models using Food-101 testing dataset, which contains 25,250 food images from 101 food categories (250 instances for each category). We measured the performance using a **classification** and a **retrieval** task discussed below:

- **Classification:** We tested the performance of the classification network in each of the models above. We adopted the standard 10-crop testing, i.e., the network made a prediction by extracting ten patches from an image (the four corner patches and the center patch in the original images and their horizontal reflections), and averaging the predictions at the softmax layer. We used Top-1 and Top-5 accuracy as metrics.
- **Retrieval:** We used a retrieval task to evaluate the quality of the learned distance embeddings. Ideally, the distances should be smaller for similar image pairs and larger for dissimilar pairs. Therefore, as suggested by previous work [173, 176], We retrieved the nearest k -neighbors of each testing image, for $k = 1, 2, \dots, N$, where $N = 25250$ is the size of the testing dataset, and calculated the Precision and Recall values for each k . We used mean Average Precision (mAP) as the evaluation metric to compare the perfor-

Table 4.3: The classification task performance. * represents the state-of-the-art approaches, and the boldface text indicates the method with the best performance.

Method	Top-1 ACC (%)	Top-5 ACC(%)
RFDC* [17]	50.76%	--
GoogleLeNet* [107]	79%	--
AlexNet+CL	67.63%	89.02%
AlexNet+MT	70.50%	90.36%
VGG+CL	82.48%	95.70%
VGG+MT (FoodDist)	83.09%	95.82%

mance. For each method, the Precision/Recall values are averaged over all the images in the testing set.

The evaluation results are summarized in Table. 4.3 and 4.4. FoodDist performs the best among four models and is significantly better than the state-of-the-art approaches in both tasks. For the classification task, the classifier built on FoodDist features achieves 83.09% Top-1 accuracy, which significantly outperforms the original RFDC [17] model and the proprietary GoogLeNet model [107]; For the retrieval task, FoodDist doubles the mAP value reported by previous work [173] that only used the AlexNet-based siamese network architecture. These results demonstrate the superior performance of FoodDist in generalization and measuring the similarities between food images with great fidelity. As shown in both tables, the advantages of FoodDist generalize across not only tasks, but also different CNN architectures.

The benchmark results demonstrate that FoodDist features possess high gen-

Table 4.4: The retrieval task performance. * represents the state-of-the-art approaches, and the boldface text indicates the method with the best performance. (Note: The mAP value that we report for Food-CNN is higher because we use pixel-wise mean subtraction, whereas the original paper only used per-channel mean subtraction.)

Method	mean Average Precision (mAP)
Food-CNN* [173]	0.3084
AlexNet+CL	0.3751
AlexNet+MT	0.4063
VGG+CL	0.6417
VGG+MT (FoodDist)	0.6670

eralization ability and the euclidean distances between feature vectors reflect the similarities between food images with great fidelity. In addition, as we can observe from both tables, the multitask learning based approach always performs better than learning with classification for both tasks no matter which CNN is used. This further justifies the proposed multitask learning approach and its advantages in incorporating both label and pairwise distance information. This makes the learned features more generalizable and meaningful in the euclidean distance space.

4.6.3 End-to-end user testing

We conducted an end-to-end user testing to validate the effectiveness of the recommendation generated by Yum-me. We recruited 60 participants via the university mailing lists, Facebook, and Twitter. The goal of the user testing is to

— Are you limiting carbohydrates? May be changed at any time. —

- ☒ No Restrictions
- ☐ Low Carb — Low grains, limits high-glycemic foods
- ☐ Paleo — No grains and legumes, limits high-glycemic foods
- ☐ Diabetic/Prediabetic — No grains, no high-glycemic foods
- ☐ Ketogenic — No grains, very low carb; high fat
- ☐ Whole30 — Specialized 30 day reset
- ☐ Low FODMAP — Specialized diet for IBS and digestive issues. Cannot be combined with other plans.

— How often do you eat meat? Note: you can exclude specific proteins lower down in the quiz —

- ☒ No Restrictions
- ☐ No Red Meat — No beef, pork, or lamb
- ☐ Pescatarian — No beef, pork, lamb, chicken, or turkey
- ☐ Flexitarian — Mostly vegetarian; includes meat or seafood occasionally
- ☐ Vegetarian — No meat, poultry, or seafood; allows dairy and eggs
- ☐ Vegan — No animal products

— Do any of the following apply to you? May be changed at any time. —

- ☐ Gluten Free
- ☐ Clean Eating ⓘ
- ☐ Dairy Free
- ☐ Kosher ⓘ
- ☐ Low Sodium ⓘ
- ☐ Pregnant/Nursing ⓘ

— Are there any ingredients you prefer to avoid? Check all that apply. —

<input type="checkbox"/> Avocado	<input type="checkbox"/> Beef	<input type="checkbox"/> Bell Peppers	<input type="checkbox"/> Chicken	<input type="checkbox"/> Cilantro
<input type="checkbox"/> Eggplant	<input type="checkbox"/> Eggs	<input type="checkbox"/> Lamb	<input type="checkbox"/> Mushrooms	<input type="checkbox"/> Peanuts
<input type="checkbox"/> Pork	<input type="checkbox"/> Raw Onion	<input type="checkbox"/> Seafood	<input type="checkbox"/> Shellfish	<input type="checkbox"/> Soy
<input type="checkbox"/> Spicy Foods	<input type="checkbox"/> Tofu	<input type="checkbox"/> Tree Nuts	<input type="checkbox"/> Other <input type="text" value="Type any ingredient..."/>	

Figure 4.11: The survey used for user onboarding at PlateJoy. (The top four questions are included.)

compare Yum-me to a widely-used user onboarding approach, i.e., a traditional food preference survey (A sample survey used by PlateJoy is shown in Fig. 4.11). As Yum-me is designed for scenarios where no rating or food consumption history is available (which is common when a user is new to a platform or is visiting a nutritionist’s office), collaborative filtering algorithm that has been adopted by many state-of-the-art recommenders is not directly comparable to our system.

In this study, we used a within-subject design in which each participant ex-

Table 4.5: The statistics of nutritional expectations indicated by 60 participants. Unit: number of participants.

Nutrient	Reduce	Maintain	Increase
Calories	30	28	2
Protein	1	44	15
Fat	23	36	1

The system used for this study was implemented as a web service. And participants were instructed to access the study website from their desktop or mobile browsers. We leveraged the web for its wide accessibility, but we could easily fit Yum-me into other ubiquitous devices, as mentioned earlier.

Participants

The most common dietary restriction chosen by 60 participants was *No restrictions* (48), followed by *Vegetarian* (9), *Halal* (2) and *Kosher* (1). No participant chose *Vegan*. Participants' nutritional expectations are summarized in Table. 4.5. For *Calories* and *Fat*, the top two goals were *Reduce* and *Maintain*. For *Protein*, participants tended to choose either *Increase* or *Maintain*. For health goals, the top four choices were *Maintain calories-Maintain protein-Maintain fat* (20), *Reduce calories-Maintain protein-Reduce fat* (10), *Reduce calories-Maintain protein-Maintain fat* (10) and *Reduce calories-Increase protein-Reduce fat* (5). The statistics match well with the common health goals among the general population, i.e., people who plan to control weight and improve sports performance tend to reduce the intake calories and fat, and increase the amount of protein.

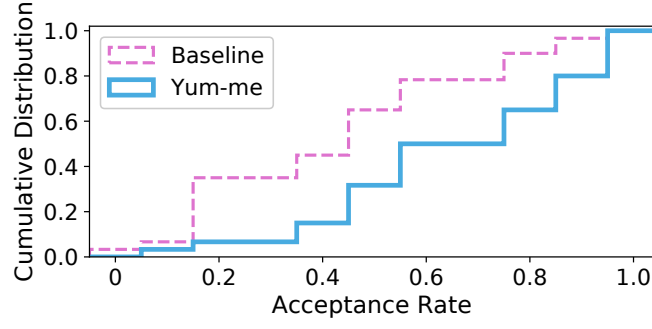


Figure 4.13: The cumulative distribution of the acceptance rate for both recommender systems.

Quantitative analysis

We use a quantitative approach to demonstrate that: (1) Yum-me recommendations yield higher acceptance rates than the baseline approach; and (2) recipes recommended by Yum-me satisfy users’ nutritional needs.

We calculated each participant’s acceptance rate of recipe recommendations as follows:

$$\text{acceptance rate} = \frac{\# \text{ of recipes in the Yummy bucket}}{\# \text{ of recommended recipes}}.$$

The cumulative distribution of the acceptance rate is shown in Fig. 4.13. And the average acceptance rate, Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) of each approach are presented in Table. 4.6. We also calculated the difference between the acceptance rates of the two systems (i.e., $\text{difference} = \text{Yum-me acceptance rate} - \text{baseline acceptance rate}$) and showed its distribution in Fig. 4.14. Yum-me outperforms the baseline by 42.63% in terms of the acceptance rate. However, we also observe that there are 12 users (20%) with zero acceptance rate difference (Fig. 4.6), which may due to the following two

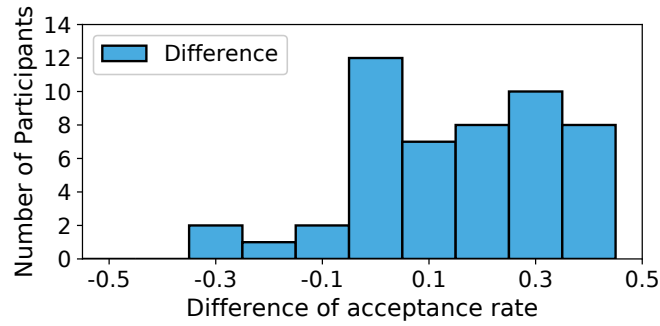


Figure 4.14: The distribution of the acceptance rate difference between two recommender systems. The difference is normally distributed (A Shapiro Wilk W test is not significant ($p = 0.12$)), and a paired Student’s t-test indicates a significant difference between the two methods ($p < 0.0001$).

Table 4.6: The Average Acceptance Rates (Avg. Acc.), Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) of two recommendation systems. Paired t-test p-value (Avg. Acc.): $< 10^{-9}$.

Metric	Mean	SEM
Yum-me Avg. Acc.	0.7250	0.0299
Baseline Avg. Acc.	0.5083	0.0341
Yum-me MAE	0.2750	0.0299
Baseline MAE	0.4916	0.0341
Yum-me RMSE	0.4481	0.0355
Baseline RMSE	0.6649	0.0290

reasons: (1) Yum-me is not effective to this set of users, and (2) these participants were not well involved in the study and randomly dragged items.

In terms of nutrition-wise performance, we compare the nutritional facts of a participant’s favorite recipes with those recommended and accepted recipes. Specifically, for users with the same nutritional needs and no dietary restric-

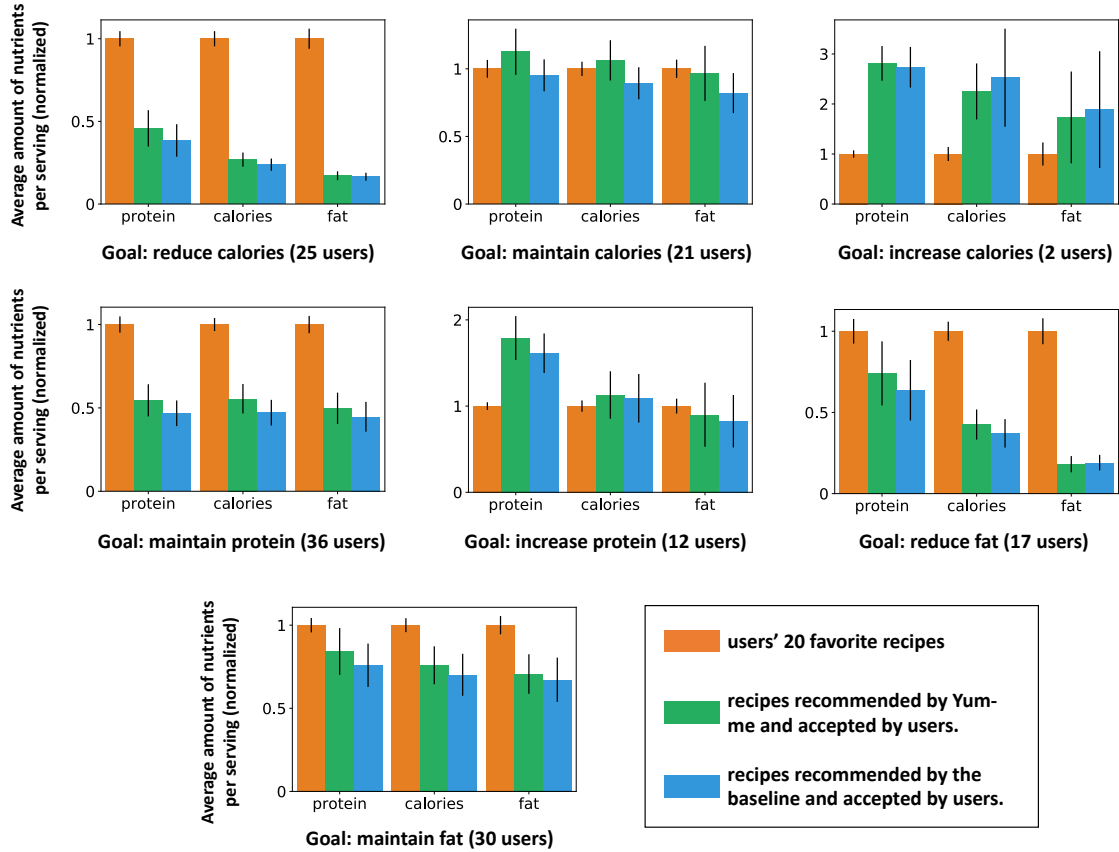


Figure 4.15: Comparison of nutritional facts among participants' favorite recipes, accepted Yum-me recommendations, and accepted baseline recommendations. The recipe is accepted if it was dragged into the *yummy* bucket. The mean values are normalized by the average amount of corresponding nutrient in the favorite recipes (orange bar). (Only 7 out of 9 nutritional goals were chosen by at least one participant.)

tions, we calculated the average amount of protein, calories and fat (per-serving) in (1) their favorite 20 recipes (as determined by our online learning algorithm), and (2) recommended and accepted recipes. The mean values presented in Fig. 4.15 are normalized by the average amount of the corresponding nutrient in their favorite recipes. The results demonstrate that Yum-me is able to satisfy most of the nutritional needs set by the participants, including *reduce*, *maintain* and *increase calories*, *increase protein*, and *reduce fat*. However, our system



Figure 4.16: A qualitative analysis of Yum-me recommendations. Images on the left half are sampled from users' top-20 favorite recipes; Images on the right half are the ones recommended to the users. The number under each food image corresponds to the amount of calories, unit: *kcal/serving*.

fails to meet two nutritional expectations, i.e., *maintain protein* and *maintain fat*. Our results also show that Yum-me recommendations can result in certain unintended consequences. For example, the goal of *reducing fat* results in a reduction in *protein* and *calories*, and the goal of *increasing calories* ends up increasing *protein*. This is partially due to the inherent dependence between nutrients. And we leave further investigation of this issue as future work.

Qualitative analysis

To qualitatively understand the recommendation mechanism of Yum-me, we randomly pick 3 participants with no dietary restrictions and aimed at reducing

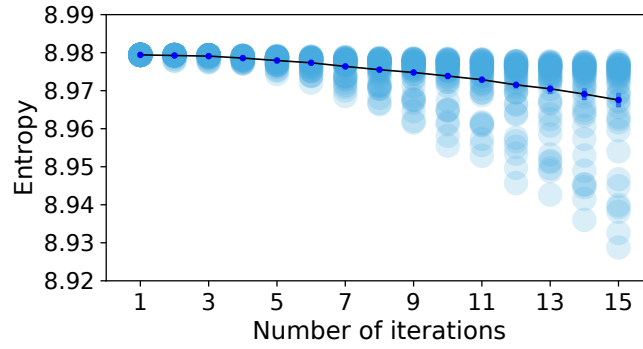


Figure 4.17: The entropy of the preference distributions in different iterations of online learning. (Using data from 48 participants with no dietary restrictions.)

calories. For each participant, we select top-20 general food items that the user liked most (inferred by the online learning algorithm). And we further select two recipes that are most similar to the ones that Yum-me recommended to the user. As shown in Fig. 4.16, our system is able to recommend recipes that are *visually similar* to the ones that the user liked, but with lower calories due to the use of healthier ingredients or different cooking styles. These examples showcase how Yum-me can leverage users’ general food preferences to rank the nutritionally appropriate items, and find the ones that are most appealing.

Error analysis

We also measured the entropy of the learned preference distribution \mathbf{p} ⁵ and find that it is negatively correlated with the improvement of Yum-me over the baseline ($r = -0.32, p = 0.026$). This correlation suggests that when a user’s preference distribution is more concentrated, the recommendations tend to be more accurate. This may due to the fact that the entropy of the preference distribution

⁵Entropy of preference distribution: $H(\mathbf{p}) = -\sum_i p_i \log p_i$

roughly reflects the degree of confidence that the system has in the user's preferences. And the confidence is higher if the entropy is lower and vice versa. In Fig. 4.17, we show the evolution of the entropy value as users are making more comparisons, which demonstrates that the system becomes more confident with more user feedback.

4.7 Discussions

In this section, we discuss the limitations of this research and present real world scenarios where **Yum-me** and its sub-modules can be used.

4.7.1 Limitations of the evaluations

When evaluating the online learning framework, we construct the baselines by combining methods that intuitively fit *user state update* and *images selection* modules. This introduces potential biases in baseline selections. Additionally, in the end-to-end user testing, the participants' judgements of whether the food is *Yummy* or *No way* is potentially influenced by the image quality and the health concerns. These may be confounding factors in measuring users' preferences towards food items and can be potentially eliminated by explicitly instructing the participants to not consider these factors. We leave further evaluations as future work.

4.7.2 Limitations of Yum-me in recommending healthy meals

The ultimate effectiveness of Yum-me in recommending healthy meals is contingent on the appropriateness of the nutritional needs input by a user. In order to conduct such recommendations for people with different conditions, Yum-me could be used in the context of personal health coaches, nutritionists or coaching applications that provide reliable nutritional suggestions based on a user's age, weight, height, exercise and disease history. For instance, general nutritional recommendations can be calculated using online services built on the guidelines from National Institutes of Health, such as *weight-success*⁶ and *active*⁷. Also, the current prototype of Yum-me assumes a relatively simple strategy to rank the nutritional appropriateness, and is limited in terms of the available options to express nutritional needs. Both issues should be addressed by future work.

4.7.3 Yum-me for real world dietary applications

We envision that Yum-me has the potential to power many real-world dietary applications. Some examples are: (1) **User onboarding.** Traditionally, dietary application (e.g., Zipongo and Plated) address the cold start problem by asking each new user to answer a set of pre-defined questions (Section 4.6.3) and then recommend meals accordingly. Yum-me can enhance this process by eliciting user's fine-grained food preference. Service providers can customize Yum-me to serve their own businesses and products by using a specialized backend food item database, and then use it as a step following the general questionnaire. (2) **Nutritional assistants.** While visiting a doctor's office, patients are often asked

⁶<http://www.weighing-success.com/NutritionalNeeds.html>

⁷<http://www.active.com/fitness/calculators/nutrition>

to fill out standard questionnaires to indicate food preferences and restrictions. Patients' answers are then investigated by the professionals to come up with effective and personalized dietary suggestions. In such a scenario, the recommendations made by Yum-me could provide a complementary channel for communicating the patients' fine-grained food preferences to the doctor for further tailored suggestions.

4.7.4 FoodDist for food image analysis tasks

FoodDist provides a unified model to extract discriminative features from food images. The model is efficient to execute ($< 0.5s/f$ on 8-core commodity processors) and can be ported to mobile devices with the publicly-available `caffe-android-lib` framework⁸.

FoodDist model can be used to fuel other nutritional applications: (1) **Food/meal recognition**. Given a set of labels, e.g., food categories, cuisines, and restaurants, the task of food recognition could be approached by first extracting features from FoodDist and then training a linear classifier, e.g., logistic regression or SVM, to classify images against categories beyond the ones given in the Food-101 dataset. (2) **Nutritional Facts estimation**. With a large-scale food image database, the problem of estimating nutritional facts might be converted to a simple nearest-neighbor retrieval task: given a query image, an algorithm retrieves its closest neighbor using features extracted from FoodDist, and then return that neighbor's nutritional information [107].

⁸<https://github.com/shlr0/caffe-android-lib>

4.8 Conclusions

This chapter presented **Yum-me**, a novel nutrient-based recipe recommender that recommends recipes catering to users’ fine-grained food preferences and nutritional needs. We also presented an online learning algorithm to efficiently learn food preferences, and **FoodDist**, a best-of-its-kind unified food image analysis model. The user study and benchmarking results demonstrated the effectiveness of Yum-me and the superior performance of FoodDist model. Looking forward, we envision that the idea of using visual similarity for preference elicitation may have implications for the following research areas: (1) **User-centric modeling**: the fine-grained food preferences learned by Yum-me is a general dietary profile that can be used to power many other dietary applications, such as suggesting meal plans for diabetes patients. Moreover, a personal dietary API can be built on top of this profile to enable cross-platform sharing. (2) **Food image analysis API for deeper content understanding**: many dietary applications, in particular the ones that capture a large number of food images, might benefit from a deeper understanding of their image contents. For instance, food journaling applications can benefit from the automatic analysis of food images to summarize the day-to-day food intake or trigger contextual reminders and suggestions. (3) **Fine-grained preference elicitation leveraging visual interfaces**. The idea of eliciting users’ fine-grained preferences via visual interfaces is applicable to other domains. Visual content captures many subtle variations missing in text and categorical data. And an adaptive visual interface can learn users’ preferences in a much shorter period of time and potentially provide more pleasant user experiences than traditional approaches.

CHAPTER 5

INTERACTIVE PREFERENCE LEARNING: AN INTENTION-INFORMED SPOKEN WORD CONTENT RECOMMENDATION SYSTEM

5.1 Introduction

The previous chapter showed how recommenders can be designed to promote dietary choices aligned with people’s nutritional expectations. However, such a possibility also raises an important concern that recommendations may shift users’ content consumption relative to what they would otherwise have chosen or aspire to choose. This is because users’ choices are often sub-optimal and focus on the short-term [109], and these immediate choices then get reinforced by recommendation systems that expose users to biased sets of items. The bias of item presentations mainly comes from two sources: (1) recommenders often hold a partial and skewed view of users’ preferences that are learned from observational interaction records [172, 131], and (2) recommenders are typically subject to popularity bias [172], which hinders the system from presenting relevant items. When subject to regular exposure to these biased item sets, users’ original intention-related choices may be altered — on the one hand, users may explore more content, on the other hand, they may end up consuming trendy but mediocre or irrelevant content with low utility to them.

Prior recommendation systems literature was focused on *how many* [72, 133] and *what* [153, 50] items people choose but rarely addressed *why* people choose them. For example, are the choices a result of people’s original intentions or their interactions with recommendation systems? In other words, how recommendations may change users’ consumption from what they might have cho-

sen, or aspire to choose? These under-explored questions are critical for recommender systems to listen to users and support users' needs, intentions, and desires [87, 43].

This chapter investigates the above mentioned questions, specifically, *how intention informed recommendations modulate users' choices, as compared to intention agnostic systems?* To answer this question, we designed a randomized controlled field study [88] in the domain of podcasts, where we leveraged the **topics of interest** as an indicator of user intentions. The field study is a 2×2 experiment where two factors are two stages of app usage, and two interventions within each factor are different recommendation algorithms. First, during onboarding, users expressed their topics of interest and subscribed to a set of podcast channels through a website, where we compared a popularity-based recommender to a recommender that takes into account users' intentions (**intention-aware recommender**) in presenting channel candidates. Then, during the remainder of their participation (app usage in the field), users used a customized commercial mobile app without constraint. During this stage of the study, we compared a subscription-based recommender to a Collaborative Filtering (CF)-based recommender in populating the home feed that users interacted with everyday. Finally, participants were invited to complete a post-study survey where they gave ratings in terms of four aspects of satisfaction.

We choose podcasts as the study domain for two main reasons. First, traditional podcast content consumption is typically based on subscriptions and therefore clearly relates to user intentions — users subscribe to RSS feeds of the channels they plan to listen to and then regularly consume released episodes from those channels. Second, recommendation systems for podcasts is of grow-

ing importance but currently under-explored (Section 5.2.4).

We conducted the study with 105 urban college students, which consists of 52.5 hours of one-by-one onboarding, four weeks of field experiments with daily communications and weekly reminders, and a follow-up survey with each participant. Our key findings include:

- **Effects of onboarding recommendations:** Compared to commonly used popularity-based ranking of channels, intention-aware recommendations for user onboarding significantly raised the ratio of channel subscription and episode listening that were aligned with users' topic-wise aspirations (improvements: 72.1% and 36.5% in terms of subscriptions at onboarding and in the field, and 24.9% in terms of listening time).
- **Effects of field recommendations:** Home feeds that were populated by the CF-based recommendations significantly increased the ratio of episode listening to not-subscribed channels by 127.5%, as compared to the traditional home feeds that were filled purely with episodes from subscribed channels.
- **Interaction effects:** User satisfaction was jointly affected by the recommendation algorithms used in the two stages — the CF-based recommender improved satisfaction for users onboarded with the intention-aware recommender, whereas for others, the CF-based recommender was shown to have negative effects.

These findings suggest that recommendations can implicitly but significantly modulate users' intention-related choices — they can encourage or discourage users to pursue their aspirations and intentions. The positive modula-

tion effects can be leveraged to support healthy behavior and benefit an individual’s aspired long-term growth, as discussed in Section 5.6. Also, our study suggests a hybrid form of recommender for podcasts and subscription-based media, consisting of an intention-aware recommender for onboarding and a CF-based recommender for home feed generation. Together, these recommenders support user aspirations, encourage content exploration, and provide satisfying user experiences.

Through our study, we also find that signals regarding the utility of user engagement is not reflected in intention-agnostic statistics (e.g., total listening time and total number of subscriptions) that are commonly employed to understand user experiences (Section 5.4.1). This highlights the importance of using metrics conditioned on individual intentions to complement the understanding of recommendation effects (Section 5.6.5).

5.2 Related work

This chapter builds on and contributes to four lines of research: (1) studying the effects of recommendations, (2) investigating user intentions in using intelligent systems, (3) building recommendation systems beyond optimizing for accuracy, and (4) analyzing and leveraging spoken word content on the web.

5.2.1 Effects of recommendations

Recommendation systems were shown to increase traffic and user engagement [133], but it was recently recognized in the research community that

they can also significantly affect end users' behavior and the structure of a society. Prior work in studying the effects of recommendations mainly focused on the social network structures [142, 34, 141] and the filtering bubble problem [116, 10, 50, 70, 112, 18]. For example, the former line of research demonstrated that introducing friend-based recommendations into social network platforms exacerbates popularity bias (i.e., rich gets richer) [142] and establishes an algorithmic ceiling for minority groups of users [141]. The latter line of research illustrated how recommendations affect users' information exposure by either limiting users' information exposure to a biased scope [116, 50] or enabling users to explore ideologically diverse opinions [10, 50]. As a result, consumers and users may be fragmented [70]. Most recently, Chaney et al. [25] used a simulation to show that recommendations may lead to a homogenization of users' choices. For contextual-aware recommendations, prior work has raised the concern about their potential alternation of users' content consumption context [4].

Although prior research has revealed significant effects of recommendations in the global and individual levels, these effects are user intention-agnostic and are measured and interpreted from system designers' and experts' perspectives. It is unknown whether recommendations' effects are aligned with or deviated from users' own intentions. Our study measures effects from users' angle and contributes findings that are critical to the future user-centric recommendation systems [72, 87, 43].

5.2.2 User intentions

Understanding and leveraging user intentions is an important theme in designing intelligent systems. For example, in the context of web search, previous research [127, 95, 150, 166, 38] discovered diverse user intents in using search engines [150], e.g., for the same query, users may look for different information. The understanding and prediction of users' intents is an essential component for personalized search experience [150, 166, 38]. In other domains, such as arts and fashion [28], and psychology [44, 126], user intentions were also investigated and were shown to be predictable from behavior logs [28]. In the context of recommendation systems, prior work leveraged interactive systems to elicit signals about user intentions, such as conversation-based [82], survey-based [177], and critique-based [27] systems. Recently, Tomkins et al. [153] presented a system that recommends appropriate products for users who intend to maintain a sustainable behavior.

However, when incorporating users' intentions, the intelligent systems were often evaluated against intention-agnostic metrics, such as click-through rate, dwelling time, etc., which do not answer the questions of how these systems alter users' choices from what they might have chosen, and how much of users' intentions were satisfied. As argued by Knijnenburg et al. [87] and Ekstrand et al. [43], future recommenders should be able to satisfy *what users want* and *what they want to achieve*. Our research takes a step further and investigates how intention informed recommenders would in turn affect users' intention-related choices, which closes the feedback loop between choices and recommenders.

5.2.3 Recommendations beyond accuracy

This chapter contributes to the increasing recognition and interests in building recommender systems for objectives beyond accuracy [87, 43, 171], such as diversity [72, 178], fairness [42], novelty [151], sustainability [153], and unbiasedness [172, 131]. These objectives were motivated by the observation that recommender systems purely optimized for accuracy may have various negative effects on end users, as reviewed in Section 5.2.1. These enable recommendations to serve users with different needs and intents. Nevertheless, similar to the limitations discussed in Section 5.2.2, prior work optimized these systems using hand-crafted or expert-designed metrics (such as categorical accuracy [42]), which may or may not be aligned with users’ intentions and goals. Our study reveals the extent to which users’ choices are related to their intentions, which can be used to inform future design of recommenders beyond accuracy.

5.2.4 Web spoken word content

We conducted the field study in the domain of spoken word content (podcasts) — an emerged channel for information and entertainment [125]. In the web community, prior research was mainly focused on building web search engines [54, 15, 59, 115, 110], which index podcast metadata and audio files so as to match given text queries to audio. However, there has been very little work addressing the podcast recommendation problem. The only work we recognized was from Tsagkias et al. [155] that predicted users’ podcast preference using hand-crafted preference indicators, which can hardly be applied in the wild because of the heterogeneity of users and content. With the interests from major

media companies to serve podcasts, research is needed to build recommenders that better expose users to content beyond passive receiving. Our study contributes a hybrid form of podcast recommender that serves users' intentions, encourages exploration and results in higher user satisfaction. This chapter also presents key guidelines for the design of podcast recommenders, which can be applied to other subscription-based media platforms as well.

5.3 Study design

Our study design included collecting consumption intentions from all participants and randomly assigning participants to four independent experimental conditions. This design allowed us to conduct within-subject comparisons to understand the discrepancy between users' content consumption and intentions, and between-subject comparisons to measure the effects of different recommendations. Specifically, our study consisted of two phases: an **one-by-one video onboarding** (30 minutes) and a **field study** (four weeks), corresponding to the prominent settings under which podcast listeners are exposed to recommendations in the wild (i.e., when they first begin to use an application, and during the daily usage). Our design for both phases of the study allowed participants to interact with recommendations naturally. During onboarding, participants were instructed to subscribe to a set of podcast channels they wanted to listen to from a ranked list of candidates; and in the field, participants were provided with a customized commercial podcast mobile app (available on both Android and IOS) to listen to podcasts naturally and without study constraints. The experiment used a full 2×2 factorial design where the two factors were recommendations made in the two study stages, i.e., onboarding (**ONB**) and field

(a) General topics selection

Please select 1 to 8 topics similar to what you want to listen to in podcasts

Arts & Literature	Literature, Design, Performing Arts, Visual Arts
Business	Careers, Investing, Management & Marketing, Business News, Shopping
Comedy	
Education	K-12, Higher Education, Educational Technology, Language Courses, Training
Food	
Fashion & Beauty	
Games & Hobbies	Video Games, Automotive, Aviation, Hobbies, Other Games
Government & Organizations	National, Regional, Local, Non-Profit
Health	Fitness & Nutrition, Self-Help, Sexuality, Alternative Health
Kids & Family	
Music	
News & Politics	
Religion & Spirituality	Buddhism, Christianity, Islam, Judaism, Spirituality, Hinduism, Other
Science & Medicine	Natural Sciences, Medicine, Social Sciences
Society & Culture	Personal Journals, Places & Travel, Philosophy, History
Sports & Recreation	Outdoor, Professional, College & High School, Amateur
TV & Film	
Technology	Gadgets, Tech News, Podcasting, Software How-to

(b) Fine-grained topics selection

Choose subtopics you aspire to listen to (optional)
If you would like fine-grained recommendations, please choose subtopics from each list

Education	Business
K-12	Careers
Higher Education	Investing
Educational Technology	Management & Marketing
Language Courses	Business News
Training	Shopping

Health	Science & Medicine
Fitness & Nutrition	Natural Sciences
Self-Help	Medicine
Sexuality	Social Sciences
Alternative Health	

Figure 5.1: The web user interface designed for participants to indicate their topic-wise intentions. Participants first select up to eight general topics they want to listen to and then optionally select fine-grained topics. The topics are defined using podcast categories in iTunes.

(FIE) recommendations, and the two interventions within each factor were specific algorithms that presented channels or episodes in different orders. Below, we describe detailed design of each phase.

5.3.1 Onboarding (ONB)

We onboarded participants one-by-one using remote video conferencing software. Participants were instructed to complete two tasks during onboarding: (1) indicate their topic-wise intentions and interests, and (2) subscribe to channels that they want to listen to in the field. Participants were directed to use a website we developed to complete both tasks.

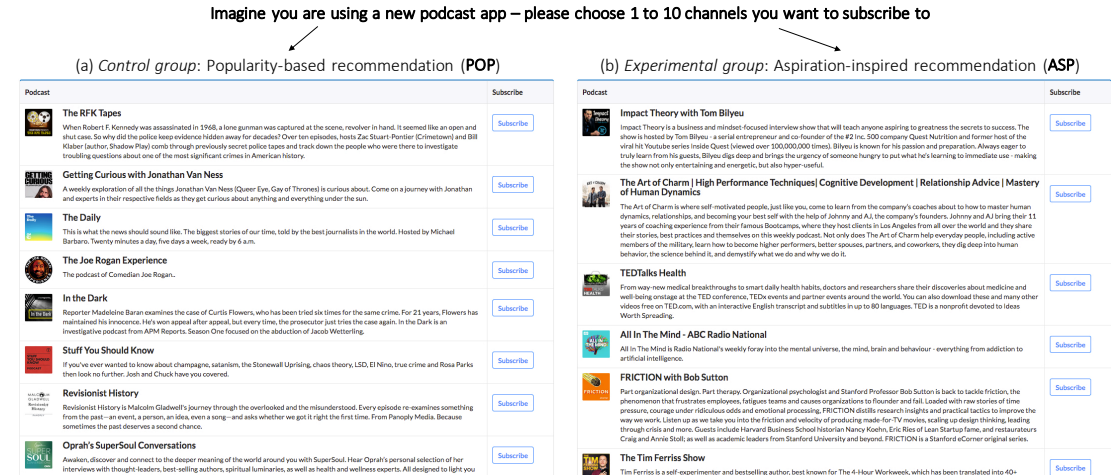


Figure 5.2: The web user interface designed for participants to subscribe to channels during onboarding. The interface presented a list of podcast shows, and participants were instructed to subscribe to up to ten of them. For the control group (POP), channels were ordered by their popularity on iTunes, whereas for the experimental group (ASP), the ordering was determined by channels' alignment to participants' topic-wise intentions. Both groups shared the same set of candidate content.

Indicating topic-wise intentions. We collected participants' listening aspirations in the form of podcast topics (Fig. 5.1). This topic selection approach is a common practice adopted by major content platforms (e.g., Pinterest and Medium) to elicit user preferences during onboarding. We used podcast categories defined by iTunes¹ as topics, which consists of two levels: general and fine-grained. Through the website, participants first picked 1-8 general topics (Fig. 5.1-a), and then optionally chose fine-grained topics within the selected general ones (Fig. 5.1-b). To help participants make sense of general topics, fine-grained topics were shown side-by-side.

Subscribing to channels. Each participant was then asked to subscribe to

¹Podcast directory: <https://itunes.apple.com/us/genre/podcasts/id26?mt=2>

up to ten podcast channels from a list of recommendations (Fig. 5.2). The recommendation list was subject to the control or experimental setting, according to the participants' assignments in the study. The control intervention implemented a standard user onboarding strategy that ordered channels based on their popularity on iTunes (**POP**) (Fig. 5.2-a), whereas the experimental intervention ranked channels by the degree to which they related to participants' aspirations² (**ASP**) (Fig. 5.2-b). The relevance of a channel c for a user u is characterized by a score $s(c|u)$ calculated as follows.

$$s(c|u) = |\mathcal{S}_c \cap \mathcal{A}_u| + \mathbb{1}[m_c \in \mathcal{A}_u], \quad (5.1)$$

where \mathcal{S}_c is the set of topics (general and fine-grained) that the channel c belongs to, m_c is the channel's primary topic ($m_c \in \mathcal{S}_c$), and \mathcal{A}_u is the set of topics that users aspired to listen to. Both \mathcal{S}_c and m_c were scraped via iTunes RSS API. As shown in the above equation, when calculating $s(c|u)$, we placed an additional weight on the primary topic.

For both groups, participants were instructed to browse the website freely and make decisions at any point of time. To prepare channels for recommendations, we scraped all top channels returned by the iTunes RSS feed, and made a join with our podcast database. Eventually, 2231 channels were used.

5.3.2 Field study (FIE)

After onboarding, each participant was provided with a podcast mobile app and a pre-registered account to use for four weeks in the field. The app was pre-loaded with the channels for which the participant subscribed during onboard-

²To break ties, channel popularity on iTunes was used.

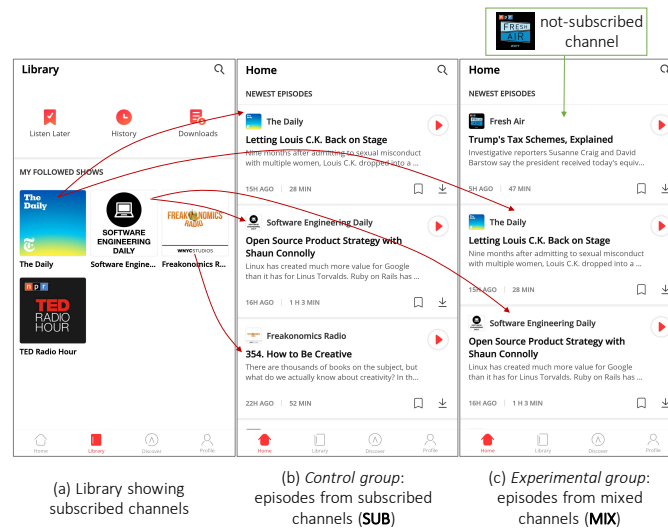


Figure 5.3: The library and home pages of the customized podcast mobile app. The library page showed the channels to which a user has subscribed, and the home page chronologically presented a list of episodes. For the control group (SUB), the episodes were retrieved from subscribed channels, whereas for the experimental group (MIX), those episodes were mixed with the ones selected from not-subscribed channels based on a CF model.

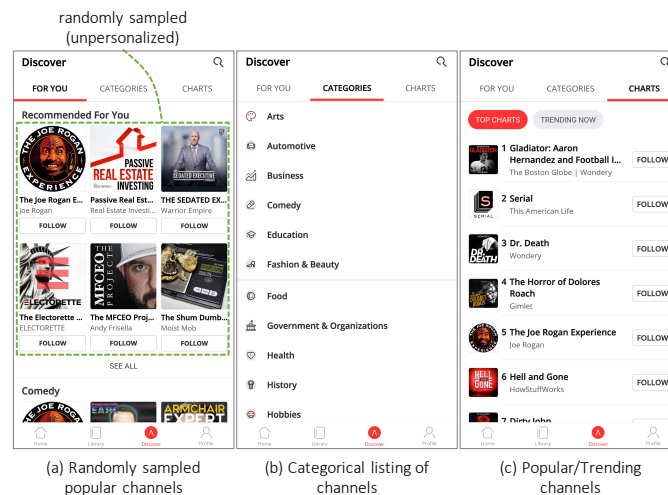


Figure 5.4: The discover page of the customized podcast mobile app. The page grouped channels into topic-wise categories and presented a trending chart that ordered channels according to their popularity on iTunes. This page allowed users to readily explore and subscribe to new channels.

ing. We customized a popular commercial app for our study. The app (shown in Fig. 5.3 and Fig. 5.4) has three main pages: (1) a **home** page (Fig. 5.3-b,c) that presented a personalized list of new podcast episodes, and is the default page when opening the app; (2) a **library** page (Fig. 5.3-a) that showed the channels to which a user has subscribed; and (3) a **discover** page (Fig. 5.4) that listed channels based on categories and popularity, which were not personalized. In addition, the app allowed users to directly search for content (through the icon at the top-right corner), and users can also consume episodes from a channel’s page (by clicking on the channel’s thumbnail).

Similar to onboarding, the field intervention was applied to recommendations on the mobile home page, which chronologically listed episodes from a personalized set of channels and was refreshed daily for newly-released episodes. For the control group, the personalized set contained channels to which a user has subscribed (**SUB**); whereas for the experimental intervention, the set additionally mixed five not-subscribed channels (**MIX**). These channels were retrieved by a matrix factorization based recommendation model, which we built as follows:

- **Dataset collection.** We scraped the most recent 500 reviews of 29K popular podcast channels on iTunes to train a recommendation model. In order to conduct recommendations based on users’ channel subscriptions, which are binary signals, we disregarded rating scores and treated iTunes reviews as positive-only feedback. The final training dataset contained 702K user-channel interactions from 137K iTunes users.
- **Recommendation model.** We used OpenRec [171] to build a Weighted Regularized Matrix Factorization (WRMF) [73] based recommender,

which is a representative implicit-feedback-based recommendation model and is optimized to minimize the following objective function:

$$\min_{x_*, y_*} \sum_{u \in \mathcal{U}, i \in \mathcal{I}} w_{ui} (p_{ui} - x_u^T y_i)^2 + \lambda \|\Theta\|^2, \quad (5.2)$$

where Θ is a set of model parameters, x_u and y_i are latent factor representations for user u (among all iTunes users \mathcal{U}) and channel i (among all iTunes channels \mathcal{I}) respectively, and p_{ui} is a binary indicator for user preferences ($p_{ui} = 1$ if user u subscribed to channel i , and $p_{ui} = 0$ otherwise). In addition, WRMF uses w_{ui} to control models' confidence levels on p_{ui} . We set w_{ui} such that $w_{ui} = 1$ if $p_{ui} = 1$, and $w_{ui} = 0.01$ otherwise. These parameter settings achieved the best validation results in our dataset. When applying the WRMF model, we discarded x_u since it corresponds to users from iTunes, and fixed trained channel representations y_i . For a participant u' , we derived an analytic expression of the optimal user representation $x_{u'}$ by differentiating the objective function (eqn. 5.2):

$$x_{u'} = \frac{1}{\lambda + \sum_i w_{u'i} y_i^T y_i} \sum_i w_{u'i} p_{u'i} y_i \quad (5.3)$$

- **Not-subscribed channel retrieval.** For any participant u' in the experimental group, we retrieved the top 5 not-subscribed channels that had the highest dot product scores (i.e., $x_{u'} y_i, i \in \{i | p_{u'i} = 0 \wedge i \in \mathcal{I}\}$). Although the recommendation model was fixed throughout the study, retrieved channels were adaptively updated as participants subscribed to new shows.

5.3.3 Post-study survey

After participants finished the 4-weeks field study, we conducted a post-study survey through email to elicit user satisfaction. The survey questions follow

Total number of participants: 105, unreported: 26	
Gender:	Female: 50 Male: 29
Age (years):	Max: 43 Min: 17 Mean: 21
Device:	IOS: 49 Android: 30
Major:	Computing and Information Science: 20 Arts & Sciences: 22 Life Sciences: 10 Medicine: 2 Business: 16 Engineering: 9

Table 5.1: Participants’ demographic information including gender, age, primary mobile device, and college major.

a template: “How satisfied were you with ___?”, and the aspects we surveyed include *the app, the experiment, your current podcast channel subscriptions, and the home feed in the app*. For each question, participants were instructed to give a likert-scale rating (i.e., *not at all satisfied, slightly satisfied, neutral, very satisfied, and extremely satisfied*).

5.3.4 Participant recruitment

We recruited 105 full-time undergraduate students who were studying in New York City and were from diverse background. The demographic information of the participants is summarized in Table. 5.1. Participants were compensated with \$30 after completing both phases of the study. To encourage app usage in the field, we provided an additional \$20 bonus for those who used the mobile app for at least five days a week, and reminded all participants to listen to new episodes weekly. Finally, participants were randomly assigned to one of the 2×2 conditions (POP-SUB: 25, POP-MIX: 26, ASP-SUB:29, ASP-MIX:25), and two research personnel who were blind to condition assignments managed and executed participants onboarding and the field study. The study was approved by the Institutional Review Board (IRB) under the protocol #1507005739.

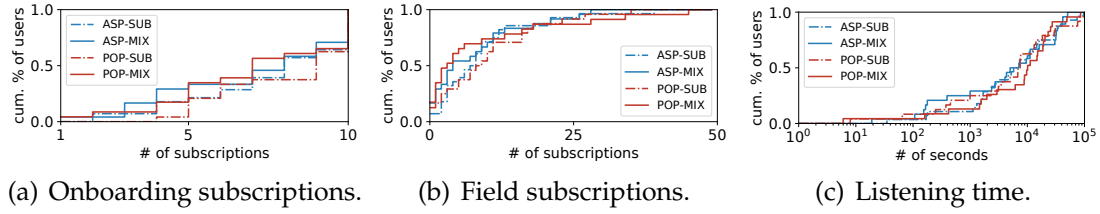


Figure 5.5: The cumulative distributions of users over the number of subscriptions and listening time. These figures show the extent to which participants were actively subscribing and listening to podcasts throughout the study. A vertical line in these figures represents a group of users with a similar activity level. We note that these commonly-used aggregated measures are not statistically different across the four groups. In other words, they do not reflect the different composition of content consumption across these groups. These differences are critical to understand the effects of recommendations on individual growth and experience.

5.4 Study results

Our study recorded the choices that participants made at onboarding and in the field including both channel subscriptions and episode listening. In addition, we recorded satisfaction ratings that participants gave to questions in the final survey. Eventually, 99 out of 105 participants completed the study (POP-SUB: 24, POP-MIX: 23, ASP-SUB:28, ASP-MIX:24). We summarize and present our study results in four dimensions: general usage patterns (Section 5.4.1), choices related to topic-wise intentions (Section 5.5.1), exploratory choices (Section 5.5.2), and user satisfaction (Section 5.5.3).

5.4.1 General usage patterns

To understand the usability and user experience with our podcast content platform, we investigate a type of commonly used metrics, user activity level [92]. We count the number of subscriptions that each user made in the field, and the amount of time that each user spent listening to episodes. The distributions of these measures over users are illustrated in Fig. 5.5. Overall, participants were fairly active in using the mobile app in the field with 8.8 average number of subscriptions and 4.58-hour average listening time. Participants' activity level is also distributed within a range and has rare outliers (Fig. 5.5-b,c). In Fig. 5.5-a, we also plot the distribution of the number of onboarding subscriptions, which is shown to spread from one to ten (maximum allowance) with an average of 7.4. To test whether two experimental factors affect the three measures in Fig. 5.5, we conduct a general nonparametric factorial analysis using the Aligned Rank Transform (**ART**) [167] (by treating the three measures as responses). We use ART because our study contains more than one factor, and all the measures are not normally distributed over users³. In the rest of this chapter, if not specified, the ART is used to conduct statistical significance tests (notations: ***: $p < 0.001$, **: $p < 0.01$, *: $p < 0.05$). The ART reports no significant effect from ONB, FIE, or ONB×FIE for all the three measures. However, as shown in Section 5.5.1 and 5.5.2, ONB and FIE have significant effects on users' podcast consumption patterns, although they are not captured in the general user activity measures. We discuss the limitations of these traditional measures in Section 5.6.5.

In addition to aggregate users' activities (subscriptions and listening time)

³The normality test is conducted via the Shapiro-Wilk normality test

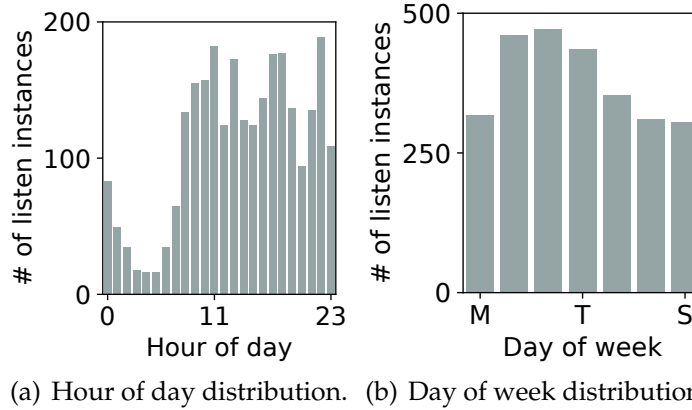


Figure 5.6: The distribution of podcast listening instances over hour of day and day of week. The aggregation is across all participants. Again we note that no statistical difference is observed across the four groups.

on a per-user basis, we also cluster the activities into hour of day (Fig. 5.6-a), day of week (Fig. 5.6-b), and distinct channels (Fig. 5.7). Temporal distributions of listening instances (Fig. 5.6) reveal several diurnal and weekly listening patterns, such as decreased listening during night and over weekends. However, no statistical evidence shows significant effects of experimental factors on these temporal patterns. Regarding the channel-wise user activity distributions (Fig. 5.7), they demonstrate that (1) during onboarding (Fig. 5.7-a), participants' channel subscriptions manifested significant popularity bias under the POP treatment, i.e., the majority of user subscriptions were concentrated on a small number of channels, whereas under the ASP treatment, subscriptions were spread out to more channels and tended to be uniformly distributed; and (2) in the field, users interacted with a broader set of podcast channels than during onboarding, but both experimental factors have no significant effect on the number of interactions that each channel received.

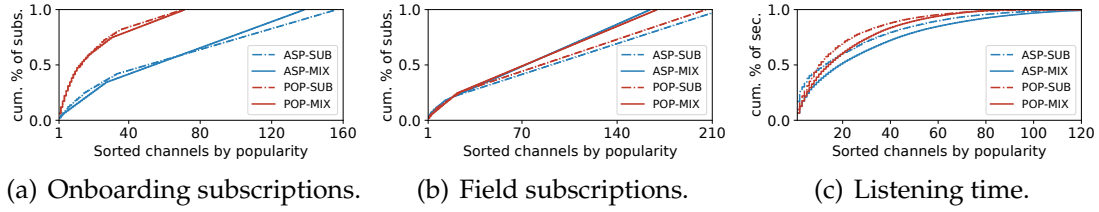


Figure 5.7: The cumulative distributions of subscriptions and listening time over channels ordered by popularity. The popularity is defined as the number of subscription (a, b) and the amount of listening (c). These figures show the extent to which participants' content consumption was concentrated on a small set of popular items. A linear line in the figure represents uniformly distributed consumption over all channels. During onboarding, the POP intervention resulted in significant popularity bias in participants' subscriptions, but in the field, no significant effect from experimental factors is observed.

5.5 Qualitative usage results

We show the channels that were most-subscribed and listened during onboarding and in the field (Fig. 5.8). During onboarding, the subscriptions made in ASP-* groups were much more diverse compared to the POP-* groups. The subscriptions from POP-* groups were mostly concentrated on trendy channels such as TED Talks Daily, TED Radio Hour, and Hidden Brain. However, in the field, all groups manifested diverse content consumption patterns, and the top subscribed and listened channels contained both trendy and long-tail items. These qualitative results further illustrate how users' podcast content consumption was driven by users' intentions and at the same time affected by recommendation systems.

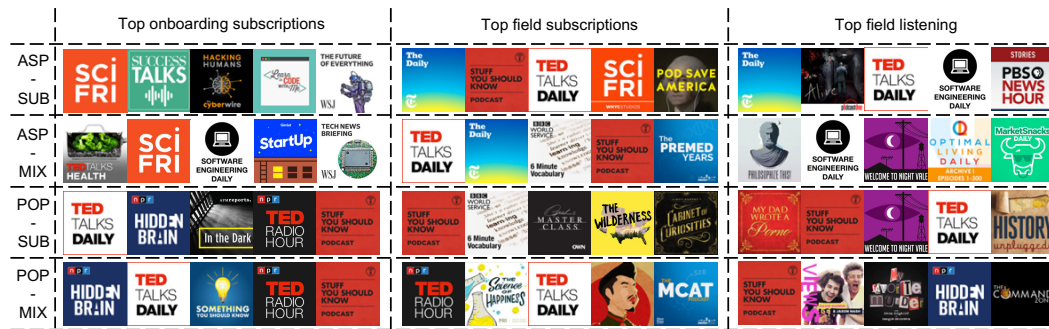


Figure 5.8: The top five most interacted content source during onboarding and in the field, categorized by 2×2 groups. Each square icon represents a podcast channel. These qualitative results demonstrate how users' content consumption in the field was jointly affected by users' intentions and recommendation systems.



Figure 5.9: The distribution of user intentions over podcast topics (categories). Topics are sorted by their popularity descendingly. Participants' intended topics were diversely spread, with most of the topics liked by less than half of the participants.

5.5.1 Choices related to topic-wise intentions

The distribution of the topics that participants intended to consume is shown in Fig. 5.9, which shows the diversity of the topics of interest chosen by participants — the intended topics in the population were spread across 53 distinct general and fine-grained categories, and most of the topics were selected by less than half of the population. Such a wide range of selected topics is partially attributable to the diverse background of our recruited participants (Table. 5.1). To show how users' choices related to topic-wise intentions may be modulated by

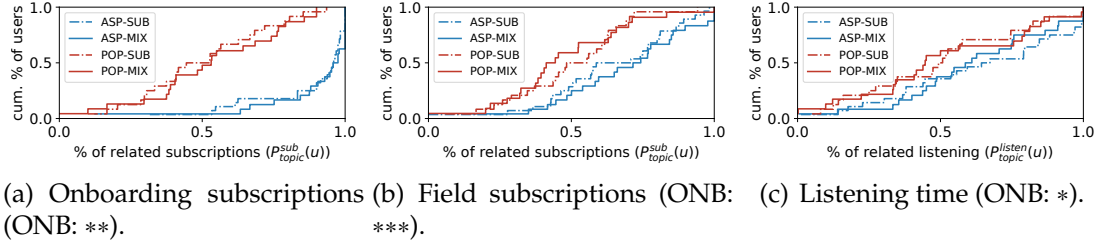


Figure 5.10: The cumulative distributions of users over the percentage of the topicwise intention-related subscriptions and listening. In the above figures, an $x = 1.0$ curve denotes that all users' consumption is related to their topicwise intentions, while an $x = 0.0$ curve denotes that none is related. The ASP intervention during onboarding is shown to significantly increase the topic-related onboarding subscriptions, topic-related field subscriptions, and topic-related field listening. The FIE factor and the interaction ONB \times FIE have no significant effect.

two stages of recommendations, we define a topic-wise intention ratio $r_{\text{topic}}(c|u)$ of a channel c for user u as follows:

$$r_{\text{topic}}(c|u) = \frac{|\mathcal{S}_c \cap \mathcal{A}_u|}{|\mathcal{S}_c|}, \quad (5.4)$$

where we use the notations from eqn. 5.1. The value of $r_{\text{topic}}(c|u)$ corresponds to the proportion of a channel's content that aligns with a user's intended topics. Then using $r_{\text{topic}}(c|u)$, we calculate the average alignment of a user u 's subscriptions, $P_{\text{topic}}^{\text{sub}}(u)$, as the average $r_{\text{topic}}(c|u)$ over all followed channels \mathcal{F}_u , i.e.,

$$P_{\text{topic}}^{\text{sub}}(u) = \frac{\sum_{c \in \mathcal{F}_u} r_{\text{topic}}(c|u)}{|\mathcal{F}_u|} \quad (5.5)$$

and calculate the average alignment of a user u 's listening, $P_{\text{topic}}^{\text{listen}}(u)$, as the weighted average of $r_{\text{topic}}(c|u)$ over listened channels \mathcal{L}_u with the weight proportional to the listening duration d_c , i.e.,

$$P_{\text{topic}}^{\text{listen}}(u) = \frac{\sum_{c \in \mathcal{L}_u} r_{\text{topic}}(c|u) d_c}{\sum_{c \in \mathcal{L}_u} d_c} \quad (5.6)$$

We show the cumulative distributions of users over $P_{\text{topic}}^{\text{sub}}(u)$ and $P_{\text{topic}}^{\text{listen}}(u)$ in Fig. 5.10, and the 2×2 groupwise averages in Fig. 5.11. These graphs and

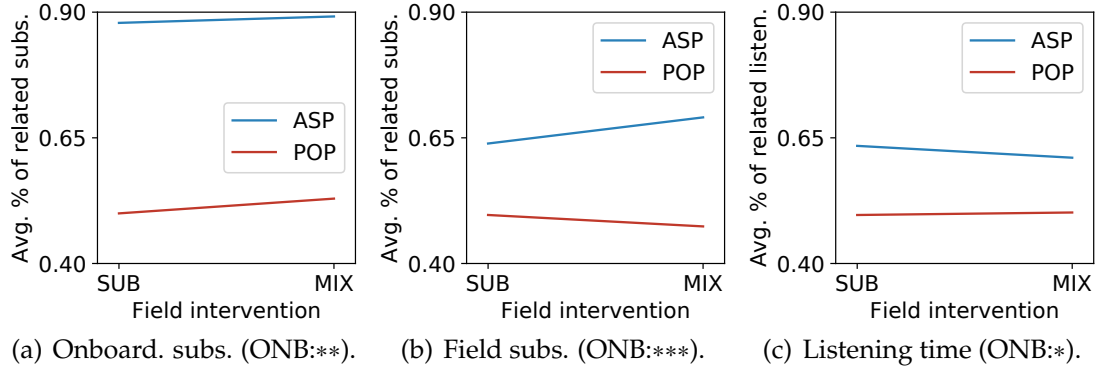


Figure 5.11: The groupwise average percentage of the topicwise intention-related subscriptions and listening. The ASP intervention significantly improves the topic-relatedness of onboarding subscriptions, field subscriptions, and field episode listening by 72.1%, 36.5%, and 24.9% respectively. The FIE and the interaction (ONB×FIE) have no significant effect.

corresponding ART tests demonstrate that under all scenarios, the ASP intervention significantly improves the ratio of content consumption that matches users’ topic-wise intentions — during onboarding, ASP increases $\bar{P}_{\text{topic}}^{\text{sub}}$ by 72.1% (ONB:**), and in the field, ASP improves $\bar{P}_{\text{topic}}^{\text{sub}}$ and $\bar{P}_{\text{topic}}^{\text{listen}}$ by 36.5% (ONB:***) and 24.9% (ONB:*) respectively. It is worth noting that although improvements are larger at onboarding when the intervention is directly applied, ASP is shown to have significant indirect effects on users’ content consumption in the field as well. The statistical test does not show significant effects from the FIE factor and the interaction (i.e., ONB×FIE).

5.5.2 Exploratory choices

To investigate how participants’ exploratory choices were affected by recommendations, we divided their podcast listening into subscribed listening (exploitation) and not-subscribed listening (exploration). We define the ex-

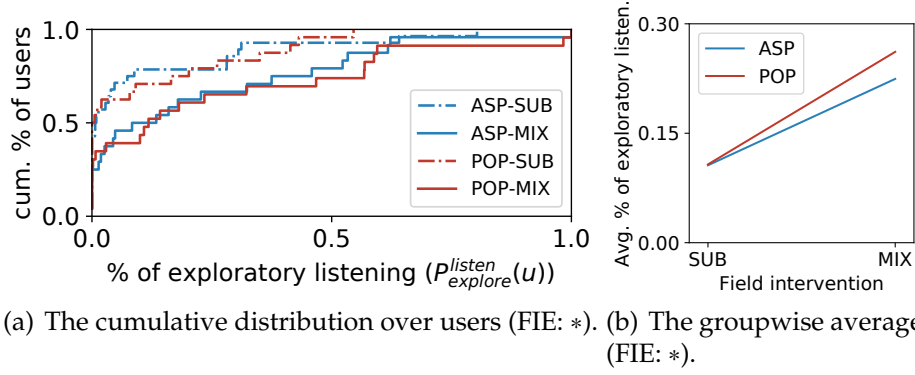


Figure 5.12: The percentage of subscriptions and listening from not-subscribed channels: (a) cumulative distributions over users, and (b) groupwise average. In (a), a $x = 1.0$ curve denotes that users do not listen to episodes from subscribed channels, while a $x = 0.0$ curve denotes that all listening comes from subscribed channels. The MIX intervention is shown to significantly increase the exploration rate by 127.5%. The ONB and the interaction (ONB \times FIE) have no significant effect.

ploratory ratio $r_{\text{explore}}(c|u)$ as a counterpart for $r_{\text{topic}}(c|u)$ (Section 5.5.1). This exploratory ratio is calculated as follows.

$$r_{\text{explore}}(c|u) = 1 - \mathbb{1}[c \in \mathcal{F}_u^t], \quad (5.7)$$

where $\mathbb{1}$ is an indicator function, and \mathcal{F}_u^t is the set of channels that user u subscribed to at time t when the channel c was consumed. Essentially, $r_{\text{explore}}(c|u) = 1$ if the channel was not subscribed when consumed, otherwise $r_{\text{explore}}(c|u) = 0$. We then substitute $r_{\text{topic}}(c|u)$ in eqn. 5.6 with $r_{\text{explore}}(c|u)$ and derive an exploratory measure of a user u 's listening, denoted as $P_{\text{explore}}^{\text{listen}}(u)$. From another angle, $P_{\text{explore}}^{\text{listen}}(u)$ can be viewed as the percentage of time that user u explored new information channels.

We show the distributions of users over $P_{\text{explore}}^{\text{listen}}(u)$ and the groupwise average scores in Fig. 5.12. Both figures and ART statistical tests demonstrate that the MIX intervention significantly increases $P_{\text{explore}}^{\text{listen}}$ by 127.5% (FIE:*). In other

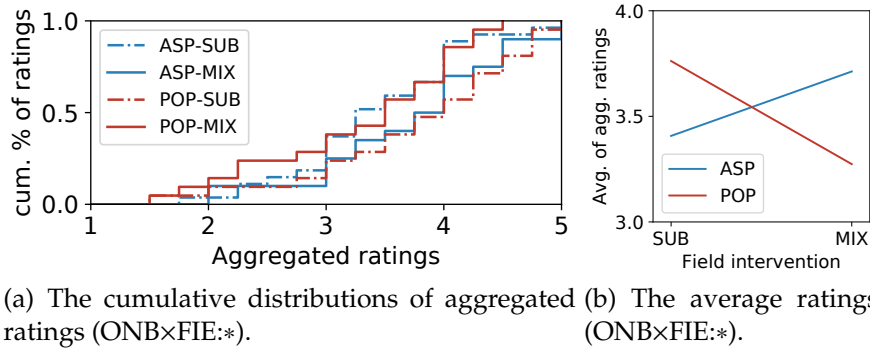


Figure 5.13: Participants' satisfaction (the averaged ratings of all indicators): (a) cumulative distributions of aggregated ratings, and (b) groupwise average ratings. The interaction between two factors (ONB×FIE) significantly affects satisfaction — MIX improves satisfaction if participants were onboarded with the ASP, otherwise MIX shows negative effects. No single factor alone has a significant effect.

words, the MIX feeds significantly encouraged participants to explore beyond existing and potentially narrow information channels. The onboarding recommendations (ONB) and the interaction between the two stages of recommendations (ONB×FIE) do not have significant effects.

5.5.3 User satisfaction

Four satisfaction indicators were surveyed and reported by participants after the study was over (Section 5.3.3). Among 99 valid participants, 89 of them responded to our email survey⁴ (Response rate: 89.9%). To quantitatively analyze survey results, following the common practice [72], we convert the five options in each survey question, i.e., *not at all satisfied*, *slightly satisfied*, *neutral*, *very satisfied*, and *extremely satisfied*, to 1–5 numerical ratings.

⁴Both experimental factors and their interaction do not have significant effects on whether or not a participant responded to the survey.

We found that satisfactions for all indicators are highly correlated. Therefore, we aggregated them into one factor by taking the average of the ratings. The distributions of the aggregated satisfaction ratings and the groupwise average values are shown in Fig. 5.13. Participants' satisfaction is significantly affected by the interaction between two factors (ONB×FIE: *); and the post-hoc differences of differences test [102, 16] confirms the effects of one factor given the other. In other words, if participants were onboarded with the popularity-based recommender, applying the CF-based recommender to populate users' home feeds significantly degraded users' satisfaction, whereas if participants were initially presented with a channel list ranked by their intentions, the CF-based recommender used in the field showed positive effects. These findings have important implications as to how the reinforcing nature of recommendations may improve or degrade utility and user experience (Section 5.6).

5.6 Implications and discussions

Our study results indicate significant interactions between recommendations and intentions. We discuss our findings in light of theoretical and empirical research on human decision making and suggest directions for designing better recommendation systems that benefit end users.

5.6.1 Employing planning and intentions

Individuals face self-control challenges when making decisions about content consumption, just as they do when managing diet or finance [109]. People

have troubles translating their intentions and goals into actions when facing real-world decision-making problems. For example, prior research showed that people rent documentaries in line with their “aspirational self” but were less likely to actually consume this type of movie compared to more affective movies such as action films [109]. Filter bubbles [116] are another example in which users’ long-term interests do not match with short-term consumption of news. To help people choose according to their long-term interests, our study suggests to employ a deliberative thinking via planning in the form of preference elicitation. As shown in Section 5.3.1, our onboarding system leveraged a preference elicitation-based interaction technique and an intention-aware recommender system that allowed for the explicit inclusion of user intentions. Such a design was shown to have significantly positive effects as users subscribed according to their elicited intentions during onboarding and later followed up on their plans when listening in the wild. Similar strategies were examined in behavioral science literature suggesting that people planning ahead are more likely to act on their intentions and to exhibit aspirational behavior in line with their long term interests [58].

5.6.2 Encouraging exploration

Classical recommendation systems based on collaborative filtering and click-based metrics are often criticized since they are likely to be overly optimized to reinforce past behavior and preferences [78]. As a result, measures such as novelty and diversity are increasingly explored both in research papers and industry practice in recent years [72, 178, 151]. Finding the right mix of novel and familiar items can be challenging as it is not clear to what extent a certain

quality characteristic like novelty is truly desired in a given application for a specific user and at a certain time. In the social and behavioral science literature this is often formulated as the exploration-exploitation trade-off [5, 77, 14]. Our results demonstrate that introducing recommendation systems in content platforms that were mainly driven by user intentions provided benefits in the form of user exploration, because recommendations helped people find choice alternatives that they were not aware of. However, how recommendation systems may influence the explore-exploit dilemma in the long term is an open question for future research.

5.6.3 Understanding user satisfaction

Our results show that users were satisfied when CF-based recommendations (MIX) were delivered based on intention-driven subscriptions (ASP) at onboarding. This can be explained by the benefits of reinforcing users' long-term interests. Whereas when users' initial subscriptions were only driven by channels' popularity and not aspirational, CF-based recommendations ignored their intentions and left them dissatisfied. Another possible explanation is the explainability and trust of recommendations. People are more likely to follow recommendations they trust, and explaining recommendations is shown to increase the trust [181]. Since the ASP-MIX hybrid recommender systems were informed by stated users' preferences, recommendations were implicitly explained and were easier to be perceived and understood by users. Whereas when the POP-MIX systems were used, the explainability and trust of field recommendations was expected to be low.

Additionally, as shown in Section 5.5.3, we also observe high user satisfaction under the POP-SUB interventions, in which users were left in their information bubble populated with self-chosen popular items. This may be explained by people’s inherent motivation to chase popular items [172] even if these items were misaligned with users’ stated intentions; such content satisfies an important, if implicit, aspect of people’s information needs and desires.

5.6.4 Optimizing for multiple objectives

Our study reveals benefits of jointly optimizing people’s information consumption for multiple objectives. For example, for podcasts and other subscription-based media, service providers should consider a hybrid form of recommender that contains an intention-aware recommender for onboarding and a CF-based recommender for field listening. This combination can support users’ intentions while encouraging them to explore beyond existing channels. As a result, users are likely to be more satisfied. More generally, with a global view of the recommendations that people are increasingly exposed to, we can jointly optimize recommendation systems to support an individual’s aspirations and satisfaction in other domains such as diet and time management.

5.6.5 Limitations of intention-agnostic metrics

Commonly-used metrics that quantify user experiences are often agnostic to people’s intentions. As a result, these metrics mainly reflect the extent to which recommendations engage people but overlook the utility of those engagements.

For example, in our study, total listening time and total number of subscriptions show that people were equally active across different groups (Section 5.4.1), but in reality, people in certain groups were less exposed to new information, guided away from their aspirations, and less satisfied. Therefore, when probing and evaluating the performance of recommendation systems, it is important to condition metrics on individual intentions.

5.7 Conclusions

This chapter presented a randomized controlled field experiment that studies the effects of recommendations on people’s content choices related to intentions. Our study revealed how recommendations (1) modulate people’s choices of topically relevant content, (2) affect the likelihood that people explore beyond their existing information sources, and (3) jointly affect user satisfaction. We discussed the implications and applications of our findings on the design, evaluation and understanding of recommendation systems. Our study confirms the suspected importance of recommendations beyond discovering relevant information. In particular, these systems implicitly alter online behavior in a manner that can have profound implications for individuals and society [4]. Future work is needed to study the generalization of these effects to wider demographic groups, and explore longer term effects of recommendations through offline evaluation, simulations, and larger scale field experiments.

CHAPTER 6

GENERALIZATION OF RECOMMENDATION ALGORITHMS

6.1 Introduction

Today's recommender systems, including user-centric recommenders presented in previous chapters, have gone beyond simple collaborative or content-based filtering algorithms to become large-scale learning machines that ingest and analyze a wide range of information. For example, diverse user feedback signals (ratings [13], click-through [73], likes [72], views [174]) and auxiliary, contextual and cross-platform traces (images [71], video [32], audio [158] and other associated metadata [119]; as well as social networks [63], software tool traces [174], and personal digital traces [72]). A state-of-the-art system [32] usually involves numerous heterogeneous and complex sub-models that analyze and fuse high-dimensional and multi-channel data streams, and each of these sub-models may have different learning architectures and a large number of hyper-parameters that need to be developed and maintained.

As a result, recommender system developers are facing an exponentially larger design space given the multiple interdependent design decisions that they need to make, such as: (1) which collaborative filtering model to use, (2) which additional data to incorporate, (3) for each additional data, which feature extraction methods to use, and (4) how to integrate the extracted features with the collaborative filtering part of the model. Moreover, researchers' design space now includes: identifying novel data sources to incorporate into the system, developing new feature extractors, and experimenting with new ways to integrate features with the user-item filtering. The software frameworks previ-

ously available for recommender systems are limited to “functional level” modularity (e.g., Librec [61] decomposes a recommender system into *inference*, *prediction*, and *similarity*), which does not provide the modularity needed to build and evaluate increasingly complex models. This chapter addresses key challenges of **extensibility** and **adaptability**.

On the one hand, traditional frameworks, such as MyMediaLite [55] and LensKit [41], usually treat a recommendation algorithm as single and **monolithic**. As a result, in order to experiment with a new method for even a small part of the algorithm, researchers often need to re-implement the whole model from scratch or extensively patch existing code. For example, to build a recommendation algorithm that incorporates image data, a researcher needs to not only implement the neural network for image analysis, but also re-build the factorization algorithm (e.g., Probabilistic Matrix Factorization), because there is no interface available in the traditional frameworks to access component modules. Significant rewriting is needed even when the recommendation is a simple composition of existing models.

On the other hand, adapting traditional frameworks to diverse recommendation scenarios requires tedious re-implementations, which may significantly affect recommendation performance despite slight implementation differences (e.g., different choices of hyper-parameters and regularization terms) [41]. We argue that such re-implementations are inevitable if the frameworks are built on diverse backend and programming languages, e.g., Java [61], C# [55], and Python [74], because of the overhead and the opportunity cost of switching between different development environments. Additionally, existing frameworks typically assume a single machine environment, which can not leverage the

computation power from distributed computing and modern hardware, e.g., GPU and TPU. Therefore, it is hard to be adopted when the model size increases.

To tackle these challenges, we propose a **modular** recommender-system design, where each recommender is a structured ensemble of reusable modules with standard interfaces. This allows the recommendation system innovation to be decomposed into (1) *designing new modules*, and (2) *inventing new computational graphs* that wire modules together. As demonstrated in Fig. 6.1, future research can readily reuse existing modules and graphs without re-implementing or modifying prior algorithms. Under such a paradigm, changes to a module or the computational graph does not affect other components, and development and testing can be more readily achieved via *plug-in and go*. Just as *modular architectures and tools* lead to rapid advances in other AI fields [30, 20, 118] and network protocol simulation [19], a modular paradigm can significantly reduce the development overhead and become fertile ground for extensible and adaptable recommender system research. In addition, we propose to use Tensorflow [1] as the standard backend for framework development. Because Tensorflow can be easily deployed in diverse computing environments (e.g., embedded devices, single machine, and distributed cloud) and is optimized for modern hardware, its use enables distributed and mini-batch (i.e., large-scale dataset) training for OpenRec and can minimize the need for language-switching re-implementations.

This chapter presents the initial design, implementation, and evaluation of **OpenRec**, an open and modular framework that supports extensible and adaptable research in recommender systems. Specifically, we build such a framework by (1) modularizing prior recommender systems, (2) identifying reusable mod-

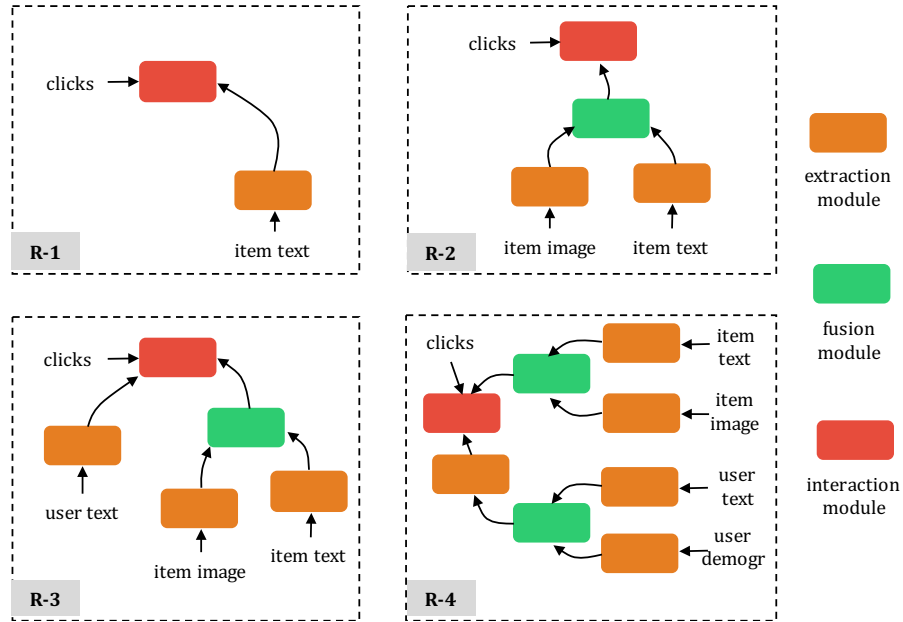


Figure 6.1: A modular view of recommendation algorithms. Each algorithm (R-1 to R-4) is a structured ensemble of reusable modules under three categories: **extraction module**, **fusion module**, and **interaction module**. The color codex is shared throughout this chapter. Arrows in the figure represent data flows.

ules and defining standard interfaces, and (3) iteratively implementing and developing in Tensorflow. In addition, we evaluate and demonstrate OpenRec in the following three contexts.

- **Reproducing monolithic implementations with OpenRec modular design.** We extensively compare the performance of the modular implementations to the prior implementations and demonstrate that the modularity in OpenRec does not degrade the models' performance in terms of both training efficiency and prediction accuracy. To the contrary, in many cases, OpenRec outperformed the existing implementations due to the ability to conduct large-batch training.
- **Rapid prototyping using OpenRec as a sandbox.** Using book recommen-

dation as an example, we illustrate how developers can use OpenRec to address specific recommendation problems by efficiently prototyping and bench-marking a large number of approaches with modules that are interchangeable.

- **Developing new recommendation algorithms by extending existing modules in OpenRec.** We use OpenRec to build a time-aware movie rating prediction algorithm for the Netflix dataset. We demonstrate that OpenRec can significantly alleviate the development burden when exploring new techniques.

The up-to-date OpenRec framework (Apache-2.0) is publicly available at:
<https://openrec.ai>

6.2 Evolution of recommender systems

In this section we briefly review the evolution of recommender systems. We discuss how recommender systems have evolved from pure collaborative filtering approaches to hybrid and content-aware models, and discuss the design challenges that arise with such development.

6.2.1 Pure collaborative filtering models

Early recommender system research focused on designing collaborative filtering models that process users' past user-item interaction data (e.g., ratings, click-through, etc.) to predict what users will like in the future [73, 13]. A well-

known example is matrix factorization, where users' past behaviors are encoded in an incomplete user-item matrix, and the prediction is made by estimating the values of the missing cells in the matrix with a low-rank assumption. Matrix factorization and other collaborative filtering models achieved great results in the Netflix competition [13], and a great amount of work has been devoted to improving upon these original approaches. The most recent examples include Neural Collaborative Filtering [68] that utilizes a neural network to allow for non-linear interactions between users and items, and Collaborative Metric Learning [71], that approaches the collaborative filtering problem from a metric learning perspective.

6.2.2 Hybrid and content-aware models

The original use cases of collaborative filtering algorithms were for the scenarios where user-item interactions are abundant (e.g., movie recommendations on Netflix or product recommendations on Amazon [99, 13]) and the user-item interaction data alone is sufficient to make high quality recommendations. However, with digital services becoming more ubiquitous in daily life, there is an increasing demand for recommender systems to work for other scenarios where users have had little or no prior interaction with the system (i.e., the user cold start scenarios), or for the scenarios where candidate items have not received much feedback from users yet (i.e., the item cold start scenarios). Collaborative filtering algorithms work poorly in such scenarios as the amount of interaction data are too sparse for them to reliably estimate users' preferences.

This demand for more powerful and diverse recommender systems, along

with the rapid advances in machine learning algorithms for content analysis, has driven a new generation of research that goes beyond the user-item matrix; in particular, new algorithms use various machine learning models to extract relevant features from additional sources [134]. For example, specific algorithms have been designed to extract item features from a large variety of signals, such as text and image data associated with items; similarly, different approaches have been proposed to extract user features from their social media traces, reviews, or other public and personal digital traces [72]. The extracted features are fused with the collaborative filtering portion of the model to allow the system to get a deeper understanding of items and users. Such hybrid models often show superior performance in cold-start scenarios, and continue to outperform the collaborative filtering solutions later on [72, 67]. Moreover, the use of content information also allows for more specific explanations to the recommendation results as compared with the generic “users like you also like this” explanations enabled by prior collaborative filtering based approaches.

6.3 Related frameworks

The rapid evolution of recommender systems (Section 6.2) has posed significant challenges to the existing software frameworks. In this section, we briefly review the limitations of existing solutions, and discuss *why OpenRec is timely and is preferable to modularizing existing frameworks*. We show the core functions of previous frameworks and their comparisons to OpenRec in Table. 6.1. Existing solutions are limited in the following two aspects.

- **Lack of algorithm level modularity support.** Previous frameworks usu-

Table 6.1: Comparing OpenRec to existing software frameworks for recommender systems (Sys-m: system-level modularity, Algo-m: algorithm-level modularity).

Framework	Sys-m	Auxiliary features	Backend	Algo-m
MLlib [137]	✗	✗	✗	✗
MyMediaLite [55]	✓	categorical	✗	✗
LensKit [41]	✓	✗	✗	✗
Surprise [74]	✓	✗	SciKits	✗
PredictionIO [24]	✓	categorical	✗	✗
Librec [61]	✓	categorical	✗	✗
OpenRec	✓	complex	Tensorflow	✓

ally provide modularity at the “functional level”, i.e., each recommender is divided into functionally-independent components (e.g., train, predict, and dataset). Such functionality-based modularity is convenient while developing new systems, but falls short when it comes to inventing and experimenting with complex algorithms, i.e., developers still need to build algorithms monolithically. In addition, because there is no “algorithm-level modularity”, it is non-trivial to add complex auxiliary features into recommendation. Therefore, OpenRec addresses a *timely* need for the recommender system community.

- **Lack of reliable backend support.** As is shown in Table 6.1, previous frameworks were built on either no explicit backend or a backend that is not scalable and unfriendly to complex models, e.g., Scikit. With such backends, the recommender systems can not leverage modern hardware, such as GPUs, and is hard to scale to distributed computing environment.

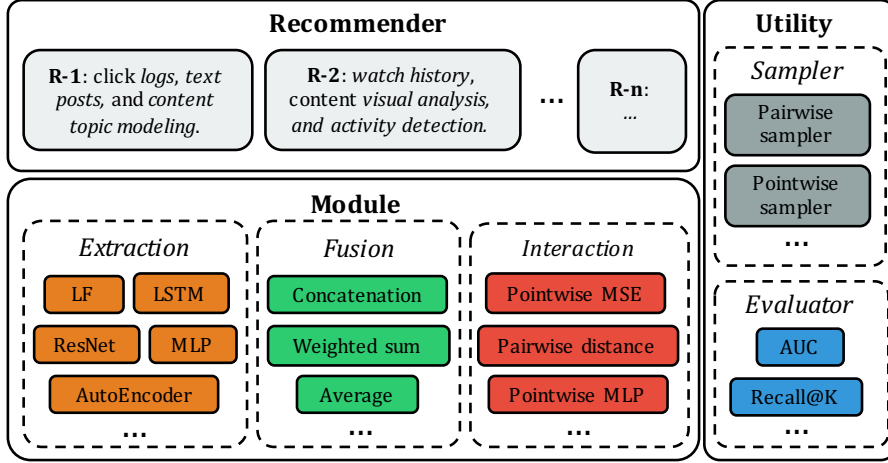


Figure 6.2: The architecture of OpenRec. A recommender is built out of modules. All three components (Module, Recommender, and Utility) can be seamlessly used together to conduct training, evaluation, experimentation, and serving of recommenders.

It is also very cumbersome for the developers to build new functions as there is little support for basic mathematical operations. Therefore, modularizing based on a legacy backend is limiting. We develop OpenRec over Tensorflow, a next generation computing engine for machine learning.

6.4 OpenRec framework

In this section, we describe the architecture of OpenRec. It views each recommendation algorithm as a computational graph that connects reusable modules together. OpenRec is comprised of two levels of abstractions - **module** and **recommender** - along with a collection of **utility** functions (Fig. 6.2). Under this framework, a **module** defines standard input/output interfaces for each category of algorithmic component. A **recommender** provides mechanisms to build end-to-end systems out of modules. **Utility** includes functions for effi-

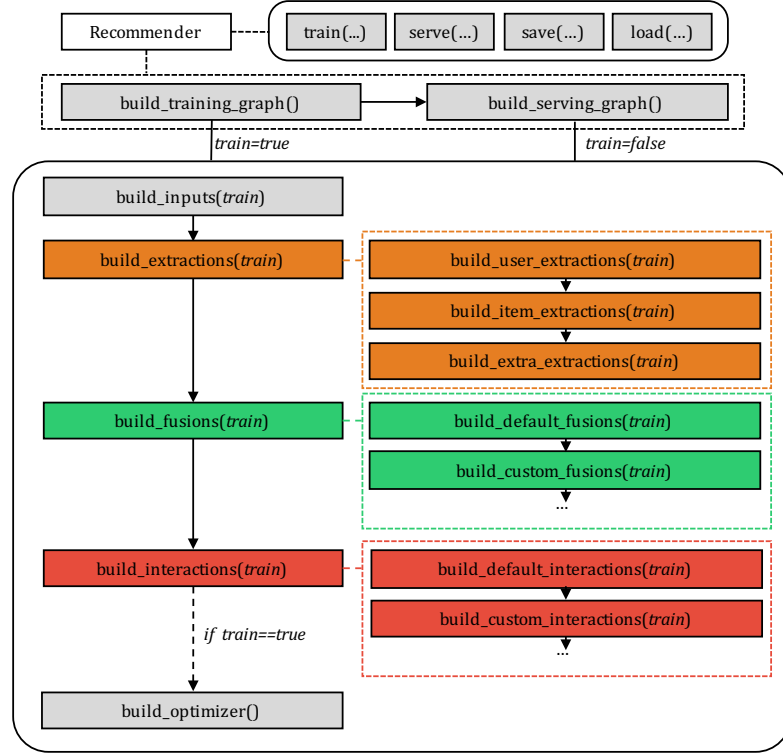


Figure 6.3: Standard interfaces of the Recommender abstraction. It contains procedures for constructing computational graphs, and functions for model training, testing, saving and loading.

cient data sampling and model evaluation. In the rest of this section, we present the details of each abstraction. Although we illustrate OpenRec with collaborative filtering approaches, the framework is also designed for more general recommendation techniques, e.g., content-based, conversational and group recommendations. We discuss the generalization of the framework in Section 6.4.4.

6.4.1 Recommenders

The Recommender abstraction provides a standard way to construct recommendation systems with modules (Section 6.4.2) and to easily conduct training and testing. The design philosophy behind the recommender is to decouple the con-

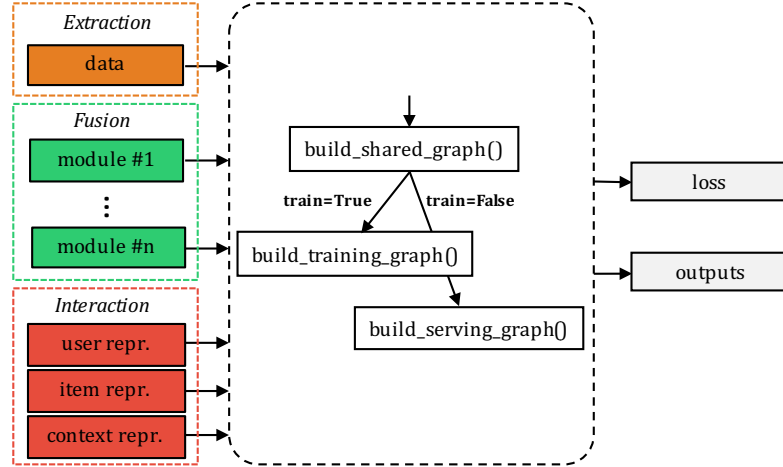


Figure 6.4: The structure of the Module abstraction (Left: inputs, Right: outputs).

struction of a complex system into many small steps, so that the system can be easily extended to include new features. As shown in Fig. 6.3, it consists of two major steps - *build training graph* and *build serving graph*, each of which calls corresponding modules. When building the training graph, a sequence of functions (i.e., *build inputs*, *build extractions*, *build fusions*, *build interactions*, and *build optimizer*) is called with the flag *train* set to *True*, where the extraction, fusion, and interaction modules are built through decomposed child functions. Similarly, when building the serving graph, all of the functions above except *build optimizer* are called with the flag *train* set to *False*. During model training, the function *train* is called for each iteration; and during testing or evaluation, the function *serve* is used to efficiently score items for a list of users. We show the flexibility and extensibility of the recommender abstraction in Section 6.5 with concrete examples.

6.4.2 Modules

Modules represent reusable components in a recommendation algorithm. As discussed in Section 6.2, a recommender typically contains three components that (1) model the interactions (including ratings, views, likes, thumb-ups, etc.) between users and items in the targeted recommendation context; (2) derive a user's, an item's or a context's representation from a data trace (Fig. 6.4), such as one-hot encoding, images, text, audio, video, location, demographic information, etc.; and (3) fuse together multiple feature representations from users, items, or environmental contexts. In OpenRec, we name components in these three categories as **interaction**, **extraction**, and **fusion** modules respectively. As shown in Fig. 6.4, OpenRec modules share the same conceptual architecture and outputs (i.e., a loss and an output list) but ingest different forms of inputs. Specifically, each module is composed of three core functions: *build shared graph*, *build training graph*, and *build testing graph*. These functions are invoked based on the value of a *train* flag that determines the mode (training or testing).

- **Interaction Module.** An interaction module takes representations from users, items or interaction contexts as inputs and then calculates the loss (during training) and item rankings (during testing). The inputs to the interaction module are typically derived from one-hot encoding or auxiliary information using extraction and fusion modules. The derived loss is used to drive the end-to-end training of the recommender system, and the item rank is used for testing and real-time recommendations. For the interaction module, we do not put any restriction on the number of users and items allowed as inputs so that it is general enough to handle a wide variety of collaborative filtering and content-based algorithms (e.g., Prob-

abilistic Matrix Factorization (PMF) is built on pairs of users and items, whereas Bayesian Personalized Ranking (BPR) requires triplets of users and items). Our initial prototype of OpenRec includes implementations of many interaction modules using state-of-the-art algorithms, e.g., pairwise logarithm used in BPR [124], pointwise mean square error (MSE) introduced by PMF [129], pairwise euclidean distance adopted in Collaborative Metric Learning (CML) [71], and pointwise cross entropy proposed by Neural Matrix Factorization (NeuMF) [68].

- **Extraction Module.** An extraction module computes representations for a data trace from users, items, or contexts. A simple example is to compute a representation from a one-hot encoding, which performs a basic lookup operation in an embedding matrix. Such a module is leveraged by traditional recommender systems without using auxiliary information, and we refer to it as a Latent Factor module. The development of extraction modules will benefit from advancements in other machine learning fields (e.g., computer vision, natural language processing and speech processing). Models from these fields can be introduced to recommender systems to analyze multi-modal data from users and items. Because OpenRec is highly modular and implemented on Tensorflow, introducing a new content analysis model is rather straightforward and efficient. In our initial prototype, we implemented two general extraction modules, Multi-layer Perceptron (MLP) and Latent Factor (LF). We expect an open source framework like OpenRec will result in development of more sophisticated models dedicated to analyzing specific data types, such as Convolutional Neural Network (CNN) for images and Recurrent Neural Network (RNN) for sequential data.

- **Fusion Module.** In many recommendation scenarios, users, items, and environmental context may have multiple data sources. For example, in the context-aware recommendation [11] and immersive recommendation [72], a user can be modeled by many personal data traces, e.g., emails, tweets and Facebook posts. To bridge the gap between multiple extraction modules and a single interaction module, an fusion module is designed to fuse multiple extraction modules together (Fig. 6.4). We prototype two intuitive fusion modules, i.e., concatenation and element-wise average.

6.4.3 Utility functions

In OpenRec, a set of utility functions are included for the ease of model training and evaluation. The model training for recommendation systems usually involves user-item sampling. For example, in BPR, (*user*, *positive-item*, *negative-item*) need to be sampled for each training batch. The samplers take the data formatted in Numpy dict as inputs and produce batches of training or validation data for a recommender. We implement popular sampling procedures (e.g., pointwise and pairwise sampling) in the OpenRec framework to drive the training process. In addition, to provide standard model testing, we implement common evaluation metrics (e.g., MSE, Recall@K and AUC) which can be seamlessly integrated with the constructed recommendation model.

6.4.4 Generalization

Since OpenRec makes few assumption about *users* and *items*, it can be used for a wide range of recommendation techniques and scenarios, e.g., recommendations with different forms of feedback, as well as interactive, conversational, and group recommendation.

- For **different forms of feedback signals**, researchers can customize sampling strategies and interaction modules, for example, using pointwise sampling with the pointwise MSE module for explicit feedback, and pairwise sampling with the pairwise logarithm module for implicit feedback.
- For **interactive and conversational recommender systems**, the optimizers can be designed to update user and item representations in the active-learning settings [177]. For every iteration, a recommender makes recommendations and updates model parameters according to users' and items' representations and their real-time interactions.
- For **group recommendations**, as OpenRec uses Numpy structured arrays as the input data format and does not have restrictions on the number of extraction modules. Users can be grouped based on the additional *group id* inputs to the sampler.

6.5 Experiments and use cases

In this section, we demonstrate the validity, efficiency and extensibility of OpenRec under the following three concrete contexts.

- **Validity.** By comparing the modular implementations with the previous ad-hoc ones, we demonstrate that modularizing recommendation algorithms does not affect the performance and efficiency. Instead, because OpenRec is built on an open-source and industry-standard deep learning tool, it can more efficiently conduct the training. (Section 6.5.1)
- **Efficiency.** Using OpenRec as a sandbox, developers are able to quickly and efficiently prototype and experiment with different settings of recommender systems and look for the optimal solution (Section 6.5.2).
- **Extensibility.** By extending and reusing existing modules, OpenRec significantly reduces the overhead of implementing **new recommendation algorithms** (Section 6.5.3).

We show the graphical illustration of the modular implementations in the main content and a sample pseudocode snippet in Section 6.5.3. More examples are available online at <https://openrec.ai>.

6.5.1 Validity: reproducing monolithic implementations

In order to test whether modularization affects the accuracy and efficiency of the recommendation algorithms, we compare the OpenRec implementations with the implementations released by the original algorithm authors. We use the same model structures and parameter settings from the original papers but replace the training strategies with the standard optimization methods adopted in OpenRec, e.g., mini-batch stochastic gradient descent. Specifically, we experiment with the following three algorithms in this chapter, each of which represents recommender systems with different complexity levels.

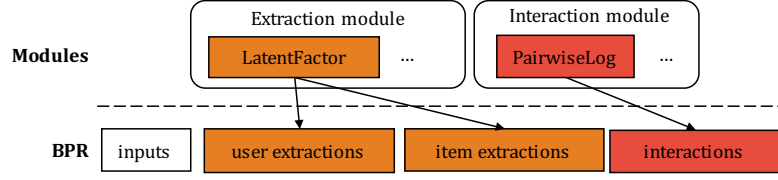


Figure 6.5: Implementing BPR with OpenRec (45 lines). We use rectangles to represent functions in a Recommender and shade the reusable modules and implementations. An arrow denotes an adoption or an inheritance. (Lines of code does not include blank and import lines.)

Bayesian Personalized Ranking (BPR)

As introduced by Rendle et al. [124], Bayesian Personalized Ranking learns latent representations for users and items by using a pairwise ranking loss (eqn. 6.1). It is one of the most popular method used under the traditional recommendation context without considering auxiliary information.

$$\sum_{(u,i,j) \in D_S} \ln \sigma(\hat{x}_{u,i} - \hat{x}_{u,j}) - \lambda_{\Theta} \|\Theta\|, \quad (6.1)$$

where $\hat{x}_{u,i} = \beta_u + \beta_i + \gamma_u^T \gamma_i$. γ represents a latent representation for a user or an item, and β denotes the corresponding bias term. D_S contains training triplets (u, i, j) where user u likes item i but does not indicate her preference for item j .

To implement the vanilla version of the BPR model using OpenRec, we employ the *Latent Factor (LF)* extraction module to compute latent representations for users and items respectively and a pairwise logarithm interaction module that takes users' and items' representations and computes the loss (Fig. 6.5). Note that we do not need to re-implement the existing modules to run the experiment. Building such a recommender system can be achieved by simply putting together the reusable modules with standard interfaces.

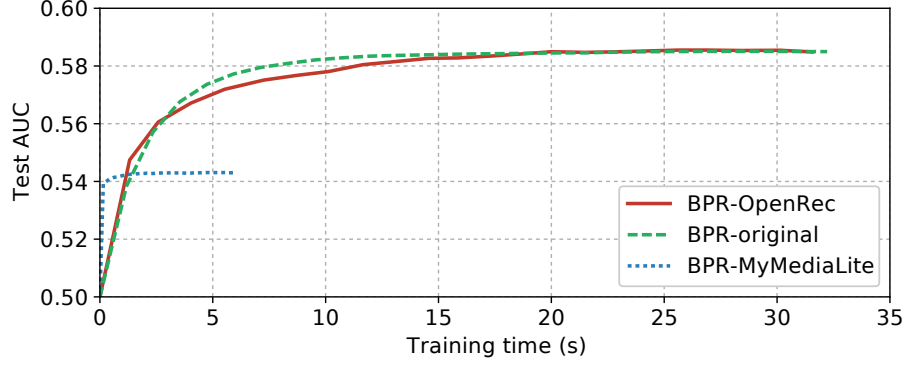


Figure 6.6: Testing performance on *tradesy.com* dataset [67] in terms of AUC (BPR-OpenRec, BPR-original, and BPR-MyMediaLite).

We compare the OpenRec modular implementation with the implementation released by He et al. [67] and MyMediaLite library [55] and evaluate them against *tradesy.com* dataset [67], where the products that users *want* and *bought* are treated as positive feedback. As a result, 19,243 users and 165,906 items are included in the experiments. For each user, we randomly sample an item that she likes for validation and another one for testing, which is consistent with the strategy used in the original paper [67]. We use the same parameter settings as [67] (λ_Θ is set to 0.1, and the dimensionality of γ is set to 20) and conduct the evaluations on an Amazon EC2 c4.4xlarge instance, which contains 16 CPU cores and 30 GB of memory.

We measure models' performance in terms of Area Under the ROC Curve (AUC), as defined in eqn. 6.2, against the training time.

$$\text{AUC} = \frac{1}{U} \sum_{u=1}^U \frac{1}{|\mathcal{P}(u)|} \frac{\sum_{(u,i) \in \mathcal{P}(u), (u,j) \in \mathcal{N}(u)} \delta(\hat{x}_{u,i} > \hat{x}_{u,j})}{|\mathcal{N}(u)|}, \quad (6.2)$$

where $\mathcal{P}(u)$ contains items user u likes in the validation/testing dataset, and $\mathcal{N}(u)$ contains items that did not receive any feedback signals from user u .

The results presented in Fig. 6.6 show that the modular implementation

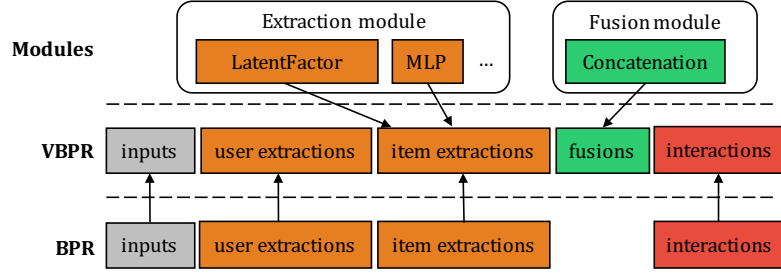


Figure 6.7: Implementing VBPR with OpenRec (50 lines). We use the same annotations as Fig. 6.5.

achieves comparable performance to the best performed BPR implementation, and significantly outperforms the implementation from previous recommendation libraries (MyMediaLite). In other words, modularization does not affect the algorithm accuracy and efficiency for simple models such as BPR.

Visual Bayesian Personalized Ranking (VBPR)

The vanilla BPR model does not incorporate any auxiliary information. To investigate recommendation scenarios where such information is leveraged, the second model that we experiment with is Visual Bayesian Personalized Ranking (VBPR), as proposed by [67]. VBPR incorporates visual features into recommendation by learning a transformation function f that projects visual features into the item embedding space. VBPR minimizes the same loss function as BPR but models $\hat{x}_{u,i}$ as follows.

$$\hat{x}_{u,i} = \beta_u + \beta_i + \gamma_u^T \gamma_i + \theta_u^T (\mathbf{E} f_i), \quad (6.3)$$

where f_i is the visual feature for item i , and \mathbf{E} is a learnable projection matrix¹.

To build such a model with OpenRec, we can easily extend the BPR recom-

¹Compared to the original VBPR, we did not include the visual biases

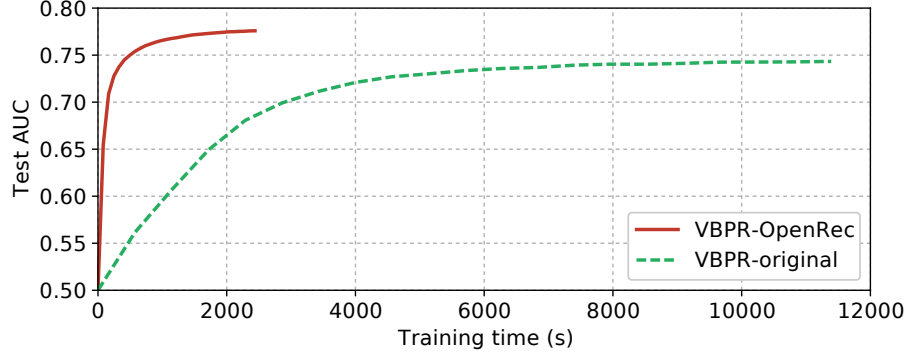


Figure 6.8: Testing performance on *tradesy.com* dataset [67] in terms of AUC (VBPR-OpenRec and VBPR-original).

mender and modify the functions *build inputs*, and *build item extractions*. We change the *build item extractions* function from a LF extraction module to a concatenation fusion module that takes as inputs the representations derived from LF and MLP extraction modules, as shown in Fig. 6.7. At the same time, other functions can be directly reused except adding additional inputs for visual features. We evaluate the VBPR implementation with the same *tradesy* dataset and computing environment, but set λ_θ to be 0.1 and the dimensionality of γ and θ to be 10. The items' visual features are extracted using the *caffe* reference model as released by the He et al. [67].

As shown in Fig. 6.8, compared to the previous implementation by He et al. [67], the model implemented by OpenRec is significantly faster (more than 10^3 times) and yields better performance in terms of AUC. The reason for such a phenomenon is that the prior implementation uses a batch size of 1 for the training while OpenRec is able to use much larger mini-batches (batch size is set to 1000) and fully utilize the available hardware resources, e.g., multi-core and GPU, using Tensorflow (We did not use GPU in the experiments for fair comparison). Under the scenario where much auxiliary information is incorporated, the larger batch size brings significant benefits, and OpenRec makes such

Table 6.2: Testing performance on citeulike dataset [162] in terms of AUC and Recall@K (CDL-OpenRec and CDL-Original).

Implementation	AUC	R@10	R@50	R@100
CDL-OpenRec	0.923	0.107	0.246	0.343
CDL-Original	0.918	0.099	0.248	0.349

benefits easily available to the end developers. This example also indicates the need for a benchmarking platform like OpenRec, as directly comparing the performance reported in the literature may be problematic, especially in the cases where some ad-hoc implementation details make significant changes to the recommendation performance.

Collaborative Deep Learning (CDL)

The third algorithm that we explore is Collaborative Deep Learning (CDL), an algorithm built upon the framework of Probabilistic Matrix Factorization (PMF) that uses a de-noising auto-encoder to incorporate text into the recommendations [162]. We refer readers to the original paper [162] for the technical details. Similar to VBPR, CDL can be implemented by extending the PMF recommender, and the extension is analogous to the Fig. 6.7. We evaluate CDL implementations on the citeulike dataset [162], which contains 5,551 users and 16,980 items and extracts item features using bag-of-words approach. We leverage the same strategy as used in the original paper to split the data into training and testing. The evaluation is conducted under the optimal parameter settings suggested by [162] and on a desktop machine with 8 CPU cores and 16 GB of memory. The performance of each implementation is measured by AUC and Recall@K after convergence. As shown in Table. 6.2, the results stay consistent with the find-

ings in the previous BPR and VBPR examples - the modular implementation of OpenRec does not degrade the performance and can completely reproduce the results from the original implementations.

6.5.2 Efficiency: quick prototyping and experimentation

In this section, we show that because of its modular nature, OpenRec can be used as a sandbox for quick designing, prototyping and evaluation in recommendation system research and development. We demonstrate this in the context of building a book recommendation system with rich context and content information, where much information is available from many different channels, including users' purchasing histories, books' content, metadata, user reviews and cover images. Therefore, developers not only need to decide what information to include in the recommender system, but also need to choose appropriate algorithms to analyze data with different modalities. In the rest of this section, we first describe the dataset for experimentation and then show the power of OpenRec in assisting and accelerating such a prototyping process.

Amazon book recommendation dataset.

We conduct experiments using an Amazon book recommendation dataset derived from an Amazon review data dump released by [106, 105]. The goal of the system is to recommend books that users are willing to buy. In this experiment, we focus on the utilities of three data sources - users' book purchasing history, users' purchases outside of book category and books' cover images. We include users who have at least 2 purchases in the book category and 5 purchases in non-

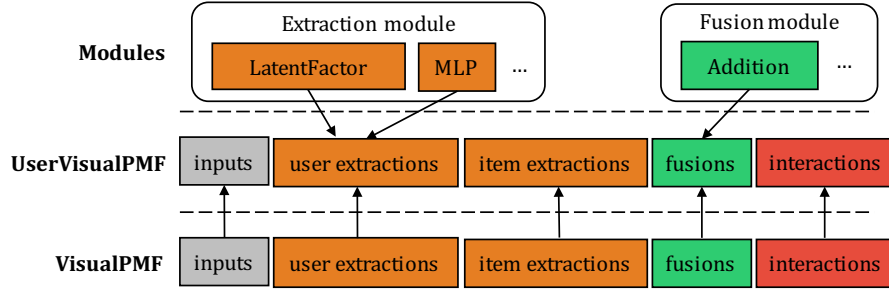


Figure 6.9: Implementing UserVisualPMF with OpenRec (32 lines). We use the same annotations as Fig. 6.5.

book categories, which ends up with a dataset containing 99,473 users, 450,166 books and 996,938 purchases. For each user, we derive a **user feature** by taking the bag-of-words representation of the labels for the products purchased in non-book categories. For each book, a **visual feature** is extracted based on the cover image using caffe reference model [79]. We divide the dataset into training/validation/testing by randomly sampling a purchase record for each user for validation and another one for testing.

What information to include?

To decide what information to include in the book recommender system, we need to experiment with combinations of the following three data sources: (A) purchasing histories, (B) user features, and (C) visual features - **PMF(A)**, **UserPMF(A+B)**, **VisualPMF(A+C)**, and **UserVisualPMF(A+B+C)**. Previously, experimenting on these models required monolithic development for each of them independently, which is a cumbersome and inefficient process. With OpenRec, the UserPMF and VisualPMF are direct extensions of PMF, and the model UserVisualPMF is an extension of UserPMF or VisualPMF. To incorporate users' or items' features, we project them into a low-dimensional embed-

ding space with a multilayer perceptron and treat the outputs as the prior for final representations. In other words, the users' or items' representations are the element-wise addition (fusion) between the projected features and the corresponding latent factors. We implement UserVisualPMF as shown in Fig. 6.9 (the implementations for VisualPMF and UserPMF are likewise). As the implementation extends most of the functions from VisualPMF and builds additional functions using reusable modules, the overhead of building UserVisualPMF is significantly reduced compared to a monolithic approach. Other fusing strategies such as concatenation are also applicable here, and OpenRec is intuitive in supporting such experiments as well. To compare the performance of these recommender systems, we select the best performed L2 regularization term among $\{0.01, 0.001, 0.0001\}$ using the validation set, and then report the AUC and Recall@K on the testing set. Because of the large number of items, for each user, we randomly sample 1000 items that did not receive any feedback signals to calculate performance metrics.

As shown in Fig. 6.10(a), in terms of AUC, adding visual features or user features significantly improves the recommendation performance, and the best performance is achieved when only visual features are incorporated. However, in terms of Recall@K, the *PMF* model performs relatively well and the *VisualPMF* is able to outperform it when $K \geq 40$. From these results, we can conclude that (1) in general, incorporating auxiliary features is helpful to book recommendations but it does not mean that more features always translate to better performance, and (2) the model selection is contingent on the metric that we want to optimize.

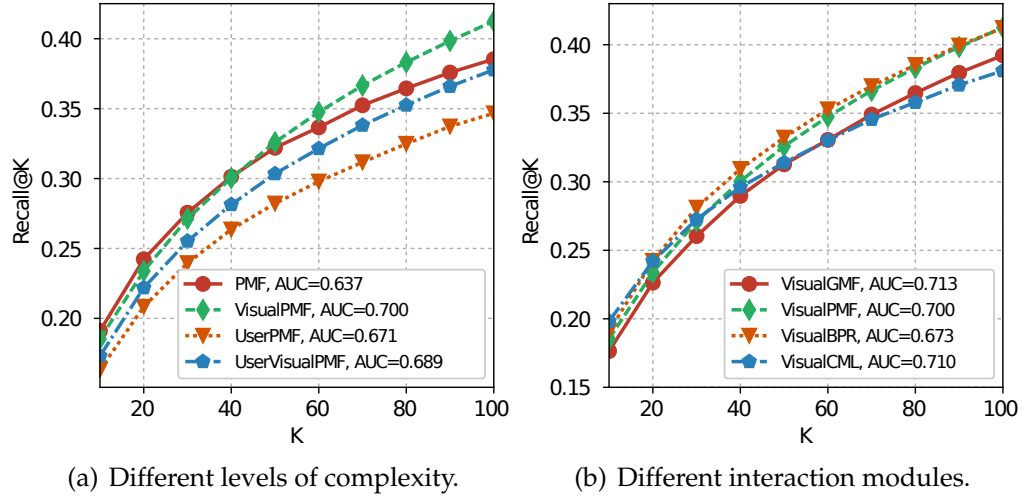


Figure 6.10: Testing performance in book recommendations in terms of AUC and Recall@K.

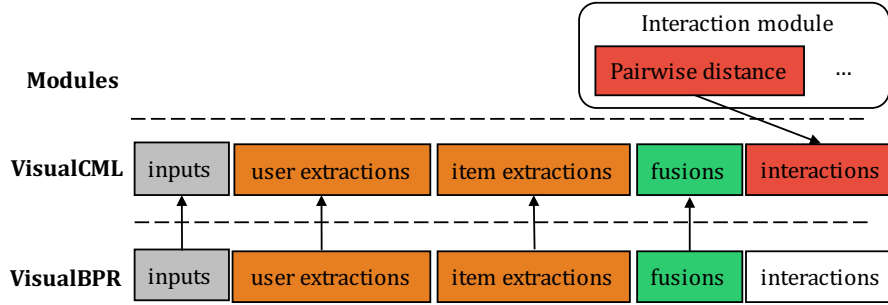


Figure 6.11: Implementing VisualCML with OpenRec (7 lines). We use the same annotations as Fig. 6.5. The model and training pseudocode are presented in Listing 1 and 2 respectively.

Which algorithm to use?

As is studied in the previous experiment, VisualPMF significantly outperforms other systems in terms of AUC. Another interesting question is whether PMF is the best collaborative filtering algorithm under such a recommendation context? We can use OpenRec to quickly investigate this question by leveraging different interaction modules and **reusing the rest of the algorithmic components**. Specifically, we show the performance of **VisualPMF**, **VisualBPR**, **VisualGMF**

```

from openrec.recommenders import VisualBPR
from openrec.modules.interactions import PairwiseEuDist

class VisualCML(VisualBPR):

    def _build_default_interactions(self, train):
        if train:
            self._interaction_train = PairwiseEuDist(train=True, ..)
        else:
            self._interaction_serve = PairwiseEuDist(train=False, ..)

```

Listing 1: Pseudocode of an OpenRec implementation for the VisualCML recommender (Section 6.5.2).

```

from openrec import ModelTrainer
from openrec.utils import Dataset
from openrec.recommenders import VisualCML
from openrec.utils.evaluators import AUC
from openrec.utils.samplers import PairwiseSampler

raw_train_data, raw_test_data = load_raw_data()
train_dataset = Dataset(raw_train_data, .., name='Train')
test_dataset = Dataset(raw_test_data, .., name='Test')

model = VisualCML(batch_size=512, ..)
sampler = PairwiseSampler(batch_size=512, dataset=train_dataset)
model_trainer = ModelTrainer(batch_size=512, dataset=train_dataset,
                             model=model, sampler=sampler, ..)
auc_evaluator = AUC()

model_trainer.train(num_itr=1e4, eval_datasets=[test_dataset],
                   evaluators=[auc_evaluator], ..)

```

Listing 2: Pseudocode of training VisualCML with OpenRec.

and **VisualCML** in Fig. 6.10(b). The sample implementation of VisualCML is shown in Fig. 6.11, which demonstrates that OpenRec provides an elegant and efficient way to quickly experiment with alternative system components.

As shown in the Fig. 6.10(b), varying the interaction module does make a difference in recommendation performance, and the best choice of the interac-

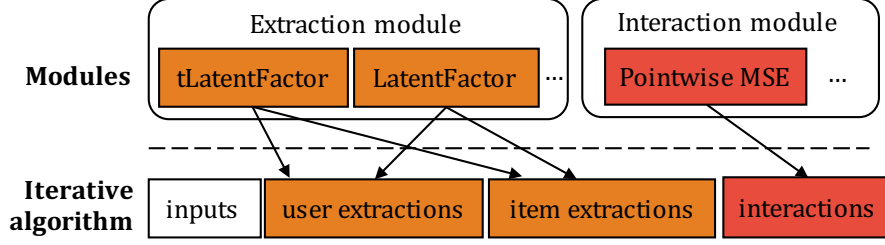


Figure 6.12: Implementing an iterative and temporal model with OpenRec (57 lines). We use the same annotations as Fig. 6.5.

tion module is dependent on the metric that we want to optimize. For example, VisualCML performs the best in terms of Recall@10, while VisualGMF achieves the best ranking performance, i.e., AUC.

The above examples also illustrate that there is no clear-cut solution to design a better recommender system. Design decisions involve trade-offs and require careful experimentation and benchmarking. With the modular design of OpenRec, we are able to support such a development process and allow experimentation of different designs with minimal overhead.

6.5.3 Extensibility: developing new algorithms via extension

In this section, we demonstrate how researchers can use OpenRec to develop new recommendation algorithms by directly extending existing modules. Specifically, we develop a light-weight, iterative and temporal recommendation model for movie rating prediction (similar to recent recommendation models [89, 168] that incorporate temporal patterns). We build multi-layer deep neural networks to project user and item vectors from time $t - 1$ to t , i.e., $\gamma_u^t = f(\gamma_u^{t-1})$ and $\gamma_i^t = g(\gamma_i^{t-1})$ where f and g are two separate multi-layer perceptron, and the most recent user and item latent representations are dot-producted to predict

the user-item ratings. To develop such a model with previous software frameworks, we would need to build everything from scratch even if there are many existing implementations of matrix factorization and multi-layer perceptrons available. However, using OpenRec, such a temporal model can be built by implementing a new extraction module *tLatentFactor* that executes the transition functions f and g and produces user and item vectors at time t , and directly extending the existing *Pointwise MSE* and *LatentFactor* modules, as Fig. 6.12 shows. This is possible because of the highly-modular nature of OpenRec. To train the model, we use traditional mean square error as the loss with L2 regularization to drive the optimization. Because OpenRec is built on a Tensorflow backend, benefits such as automatic differentiation are readily available to the developers.

We evaluate our temporal model using the Netflix dataset [13] and compare it to the traditional matrix factorization (MF) implementation from MyMediaLite. We update users' and items' representations daily² and validate on each batch before training (Each data point is only used once). The user and item vectors are initialized using MF over the first 3/4 of the dataset (75M ratings). We refer readers to the OpenRec online repository for additional parameter settings. The experimental results demonstrate that our model significantly outperforms the MF baseline by 6% in terms of MSE (0.066 for ours and 0.071 for MF) after only training on 3 days of rating data, which justifies the merits of temporal patterns. Note that our model is not intended to be the state-of-the-art in temporal recommendation but rather as an example of how researchers can easily use OpenRec to explore their ideas.

²We only make updates for users and items that have rating during that day.

6.6 Conclusions

This chapter introduced OpenRec, a modular framework designed to support extensible and adaptable development and research in recommender systems. Through careful experiments and case studies, we demonstrated the value of modularity and reusability. Moving forward, future work includes: standardizing interfaces; building new modules, recommenders, and utility functions (such as NDCG); evaluating models against standard datasets and criteria; and creating modularized models with non-neural network structures. We hope OpenRec can provide infrastructural support for broader and systematic exploration of personalization methods from consumers' and societal perspectives.

CHAPTER 7

FUTURE WORK

The work described in this thesis addressed the design, implementation and evaluation of user-centric recommendation systems. Moving forward, as intelligent systems become increasingly pervasive, future research needs to address the challenges that arise in new domains of personalization. And beyond individuals' utility, building responsible recommenders should additionally consider societal needs and balance them with commercial interests. Below are further discussions of future work.

Developing personalization algorithms and methods for intelligent assistants. Intelligent assistants (e.g., Amazon Alexa, Microsoft Cortana, Apple Siri, Google assistants, and Adobe creative assistants) recommend content, products, actions, and tools to improve productivity and creativity. Recommendations in this emerging setting introduce new research challenges and opportunities. First, user interactions with intelligent assistants are sporadic, which differs from traditional recommendation settings where users almost continuously interact with online systems. In other words, personal assistants need to model user behavior and preferences through sparse interactions. A potential approach to addressing this problem is to bootstrap user profiles from cross-platform data traces, as shown in Chapter 3. In addition, intelligent assistants are inherently interactive, which requires a new algorithmic framework to optimize the user–system feedback loop for user modeling, potentially in the form of conversations. This may significantly build upon and extend prior human-in-the-loop research. An example of such an algorithm is discussed in Chapter 4. Lastly, recommendations from personal assistants need to be context-aware (i.e.,

aware of time, location, weather, and mood). Future research should develop methods for assistants to sense and incorporate personal and environmental context. This can potentially be achieved by leveraging multimodal sensing data from mobile and wearable devices and external knowledge bases.

Personalization and recommendation as applied to education and wellness applications. Recommendation systems are integral components of personalized learning and precision health (medicine) applications. For example, recommending courses and personalizing learning paces in online education platforms; and balancing working time and suggesting exercises for better health outcomes. For these application scenarios, future research challenges include: (1) Incorporating necessary domain knowledge into personalization models. We can approach this challenge by exploring re-ranking techniques (as shown in Chapter 4) and innovative designs of objective functions that drive the end-to-end training of recommendation models. (2) Building interpretable and accountable systems — adopting health or learning advice can be costly and risky, and can even have a lifelong impact on people’s lives. Therefore, making predictive models interpretable and explaining the recommendations can help people make informed decisions. (3) Enabling user control. Algorithmic recommendations inevitably make mistakes. Enabling users to control and give feedback on recommendations easily can steer the systems when they are divergent from users’ aspirations.

Open platforms and tools for research, experimentation, and deployment. Building, deploying and investigating recommendation and personalization systems under different scenarios is complex. Implementing proposed innovations in this space from the ground up is error-prone, impedes research col-

laboration and iteration, and is often not reproducible. Built on the OpenRec library developed in this thesis (Chapter 6), future work should continue refining and innovating on open-source platforms and tools, including algorithms, simulation environments, datasets, and modular interfaces, so that the research community can build on them and practitioners can use them for real-world deployments. To make recommenders more user-centric, it is particularly important to reduce the friction associated with adoption in real-world systems.

Understanding and mitigating unintended consequences of recommendation and personalization. Personalization algorithms have profound impacts on people’s daily lives and our society. For example, in Chapter 5, we show that recommendation systems can modulate people’s content choices related to their aspirations. However, in reality, many impacts are not intended when developing personalization solutions, e.g., reduced productivity and creativity, altered views and values, biases and unfairness, risk inclination behavior [49, 75], and invasion of user privacy. It is critical to understand and mitigate these consequences as personalization techniques are getting deployed. We can approach this problem in three directions: First, applying counterfactual reasoning and causal inference on large-scale behavioral data, which logs how people’s behavior and choices change over time. This technique has been explored in Chapter 2 and is shown to produce promising results in debiasing recommender evaluation. Second, conducting field experiments to measure the effects of different recommendation interventions (as shown in Chapter 5). Third, building numerical simulation environments by leveraging human choice models. Recent work [25] has shown promising results in this direction.

BIBLIOGRAPHY

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.
- [2] Himan Abdollahpour, Robin Burke, and Bamshad Mobasher. Controlling popularity bias in learning-to-rank recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 42–46. ACM, 2017.
- [3] Eytan Adar, Mira Dontcheva, and Gierad Laput. Commandspace: modeling the relationships between tasks, descriptions and features. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 167–176. ACM, 2014.
- [4] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer, 2011.
- [5] Deepak Agarwal, Bee-Chung Chen, Pradheep Elango, and Raghu Ramakrishnan. Content recommendation on web portals. *Communications of the ACM*, 56(6):92–101, 2013.
- [6] Deepak Agarwal, Bee-Chung Chen, Qi He, Zhenhao Hua, Guy Lebanon, Yiming Ma, Pannagadatta Shivaswamy, Hsiao-Ping Tseng, Jaewon Yang, and Liang Zhang. Personalizing linkedin feed. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1651–1660. ACM, 2015.
- [7] Lenore Arab, Deborah Estrin, Donnie H Kim, Jeff Burke, and Jeff Goldman. Feasibility testing of an automated image-capture method to aid dietary recall. *European journal of clinical nutrition*, 65(10):1156–1162, 2011.
- [8] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *ACM-SIAM symposium on Discrete algorithms*, 2007.
- [9] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 2002.

- [10] Eytan Bakshy, Solomon Messing, and Lada A Adamic. Exposure to ideologically diverse news and opinion on facebook. *Science*, 348(6239):1130–1132, 2015.
- [11] Linas Baltrunas, Bernd Ludwig, and Francesco Ricci. Matrix factorization techniques for context aware recommendation. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 301–304. ACM, 2011.
- [12] Oscar Beijbom, Neel Joshi, Dan Morris, Scott Saponas, and Siddharth Khullar. Menu-match: restaurant-specific food logging from images. In *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*, pages 844–851. IEEE, 2015.
- [13] James Bennett and Stan Lanning. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35, 2007.
- [14] Oded Berger-Tal, Jonathan Nathan, Ehud Meron, and David Saltz. The exploration-exploitation dilemma: a multidisciplinary framework. *PloS one*, 9(4):e95693, 2014.
- [15] Jana Besser, Martha Larson, and Katja Hofmann. Podcast search: User goals and retrieval technologies. *Online information review*, 2010.
- [16] Robert J Boik. Interactions, partial interactions, and interaction contrasts in the analysis of variance. *Psychological Bulletin*, 86(5):1084, 1979.
- [17] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101—mining discriminative components with random forests. In *Computer Vision—ECCV 2014*, pages 446–461. Springer, 2014.
- [18] Danah Boyd. Streams of content, limited attention: The flow of information through social media. *Educause Review*, 45(5):26, 2010.
- [19] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, et al. Advances in network simulation. *Computer*, 33(5):59–67, 2000.
- [20] Denny Britz, Anna Goldie, Thang Luong, and Quoc Le. Massive Exploration of Neural Machine Translation Architectures. *ArXiv e-prints*, March 2017.

- [21] JR Brotherhood. Nutrition and sports performance. *Sports Medicine*, 1(5):350–389, 1984.
- [22] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.
- [23] Junghoon Chae, Insoo Woo, SungYe Kim, Ross Maciejewski, Fengqing Zhu, Edward J Delp, Carol J Boushey, and David S Ebert. Volume estimation using food specific shape templates in mobile image-based dietary assessment. In *IS&T/SPIE Electronic Imaging*, pages 78730K–78730K. International Society for Optics and Photonics, 2011.
- [24] 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.
- [25] Allison JB Chaney, Brandon M Stewart, and Barbara E Engelhardt. How algorithmic confounding in recommendation systems increases homogeneity and decreases utility. *arXiv preprint arXiv:1710.11214*, 2017.
- [26] Shuo Chang, F. Maxwell Harper, and Loren Terveen. Using groups of items for preference elicitation in recommender systems. In *CSCW*, 2015.
- [27] Li Chen and Pearl Pu. Critiquing-based recommenders: survey and emerging trends. *User Modeling and User-Adapted Interaction*, 22(1-2):125–150, 2012.
- [28] Justin Cheng, Caroline Lo, and Jure Leskovec. Predicting intent using activity logs: How goal specificity and temporal range affect user behavior. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 593–601. International World Wide Web Conferences Steering Committee, 2017.
- [29] Edward Choi, Mohammad Taha Bahadori, Elizabeth Searles, Catherine Coffey, and Jimeng Sun. Multi-layer representation learning for medical concepts. *arXiv preprint arXiv:1602.05568*, 2016.
- [30] François Chollet et al. Keras. <https://github.com/fchollet/keras>, 2015.
- [31] Felicia Cordeiro, Elizabeth Bales, Erin Cherry, and James Fogarty. Re-thinking the mobile food journal: Exploring opportunities for lightweight

- photo-based capture. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3207–3216. ACM, 2015.
- [32] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 2016.
 - [33] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. *arXiv preprint arXiv:1512.04412*, 2015.
 - [34] Elizabeth M Daly, Werner Geyer, and David R Millen. The network effects of recommending social connections. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 301–304. ACM, 2010.
 - [35] Mahashweta Das, Gianmarco De Francisci Morales, Aristides Gionis, and Ingmar Weber. Learning to question: leveraging user preferences for shopping advice. In *KDD*, 2013.
 - [36] 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.
 - [37] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.
 - [38] Zhicheng Dou, Ruihua Song, and Ji-Rong Wen. A large-scale evaluation and analysis of personalized search strategies. In *Proceedings of the 16th international conference on World Wide Web*, pages 581–590. ACM, 2007.
 - [39] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.
 - [40] Michael Ekstrand, Wei Li, Tovi Grossman, Justin Matejka, and George Fitzmaurice. Searching for software learning resources using application context. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 195–204. ACM, 2011.

- [41] 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.
- [42] Michael D Ekstrand, Mucun Tian, Ion Madrazo Azpiaz, Jennifer D Ekstrand, Oghenemaro Anuyah, David McNeill, and Maria Soledad Pera. All the cool kids, how do they fit in?: Popularity and demographic biases in recommender evaluation and effectiveness. In *Conference on Fairness, Accountability and Transparency*, pages 172–186, 2018.
- [43] Michael D Ekstrand and Martijn C Willemsen. Behaviorism is not enough: better recommendations through listening to users. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 221–224. ACM, 2016.
- [44] Andrew J Elliot and Judith M Harackiewicz. Goal setting, achievement orientation, and intrinsic motivation: A mediational analysis. *Journal of personality and social psychology*, 66(5):968, 1994.
- [45] David Elswailer and Morgan Harvey. Towards automatic meal plan recommendations for balanced nutrition. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 313–316. ACM, 2015.
- [46] Leonard H Epstein, Rena R Wing, Barbara C Penner, and Mary Jeanne Kress. Effect of diet and controlled exercise on weight loss in obese children. *The Journal of pediatrics*, 107(3):358–361, 1985.
- [47] Kevin J Eschleman, Jamie Madsen, Gene Alarcon, and Alex Barelka. Benefiting from creative activity: The positive relationships between creative activity, recovery experiences, and performance-related outcomes. *Journal of Occupational and Organizational Psychology*, 87(3):579–598, 2014.
- [48] Deborah Estrin. Small data, where n= me. *Communications of the ACM*, 57(4):32–34, 2014.
- [49] Peter Fischer, Evelyn Vingilis, Tobias Greitemeyer, and Claudia Vogrinic. Risk-taking and the media. *Risk Analysis: An International Journal*, 31(5):699–705, 2011.
- [50] Seth Flaxman, Sharad Goel, and Justin M Rao. Filter bubbles, echo chambers, and online news consumption. *Public opinion quarterly*, 80(S1):298–320, 2016.

- [51] Peter Forbes and Mu Zhu. Content-boosted matrix factorization for recommender systems: experiments with recipe recommendation. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 261–264. ACM, 2011.
- [52] C Ailie Fraser, Mira Dontcheva, Holger Winnemoeller, and Scott Klemmer. Discoveryspace: Crowdsourced suggestions onboard novices in complex software. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*, pages 29–32. ACM, 2016.
- [53] Jill Freyne and Shlomo Berkovsky. Intelligent food planning: personalized recipe recommendation. In *Proceedings of the 15th international conference on Intelligent user interfaces*, pages 321–324. ACM, 2010.
- [54] Marguerite Fuller, Manos Tsagkias, Eamonn Newman, Jana Besser, Martha Larson, Gareth JF Jones, and Maarten de Rijke. Using term clouds to represent segment-level semantic content of podcasts. 2008.
- [55] 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.
- [56] Gijs Geleijnse, Peggy Nachtigall, Pim van Kaam, and Luciënne Wijgergangs. A personalized recipe advice system to promote healthful choices. In *Proceedings of the 16th international conference on Intelligent user interfaces*, pages 437–438. ACM, 2011.
- [57] Nadav Golbandi, Yehuda Koren, and Ronny Lempel. Adaptive bootstrapping of recommender systems using decision trees. In *WSDM*, 2011.
- [58] Peter M Gollwitzer. Implementation intentions: strong effects of simple plans. *American psychologist*, 54(7):493, 1999.
- [59] Masataka Goto and Jun Ogata. Podcastle: Recent advances of a spoken document retrieval service improved by anonymous user contributions. In *Twelfth Annual Conference of the International Speech Communication Association*, 2011.
- [60] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. E-commerce

- in your inbox: Product recommendations at scale. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1809–1818. ACM, 2015.
- [61] Guibing Guo, Jie Zhang, Zhu Sun, and Neil Yorke-Smith. Librec: A java library for recommender systems. In *UMAP Workshops*, 2015.
 - [62] Ido Guy, Naama Zwerdling, David Carmel, Inbal Ronen, Erel Uziel, Sivan Yogeve, and Shila Ofek-Koifman. Personalized recommendation of social software items based on social relations. In *Proceedings of the third ACM conference on Recommender systems*, pages 53–60. ACM, 2009.
 - [63] Ido Guy, Naama Zwerdling, Inbal Ronen, David Carmel, and Erel Uziel. Social media recommendation based on people and tags. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 194–201. ACM, 2010.
 - [64] Morgan Harvey, Bernd Ludwig, and David Elswailer. You are what you eat: Learning user tastes for rating prediction. In *International Symposium on String Processing and Information Retrieval*, pages 153–164. Springer, 2013.
 - [65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.
 - [66] Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proceedings of the 25th International Conference on World Wide Web*, pages 507–517. International World Wide Web Conferences Steering Committee, 2016.
 - [67] Ruining He and Julian McAuley. Vbpr: Visual bayesian personalized ranking from implicit feedback. In *AAAI*, pages 144–150, 2016.
 - [68] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 173–182. International World Wide Web Conferences Steering Committee, 2017.
 - [69] Ye He, Chang Xu, Neha Khanna, Carol J Boushey, and Edward J Delp. Food image analysis: Segmentation, identification and weight estima-

- tion. In *Multimedia and Expo (ICME), 2013 IEEE International Conference on*, pages 1–6. IEEE, 2013.
- [70] Kartik Hosanagar, Daniel Fleder, Dokyun Lee, and Andreas Buja. Will the global village fracture into tribes? recommender systems and their effects on consumer fragmentation. *Management Science*, 60(4):805–823, 2013.
 - [71] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. Collaborative metric learning. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017.
 - [72] Cheng-Kang Hsieh, Longqi Yang, Honghao Wei, Mor Naaman, and Deborah Estrin. Immersive recommendation: News and event recommendations using personal digital traces. In *Proceedings of the 25th International Conference on World Wide Web*, pages 51–62. International World Wide Web Conferences Steering Committee, 2016.
 - [73] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM’08. Eighth IEEE International Conference on*, pages 263–272. Ieee, 2008.
 - [74] Nicolas Hug. Surprise, a Python library for recommender systems. <http://surpriselib.com>, 2017.
 - [75] Jay G Hull, Ana M Draghici, and James D Sargent. A longitudinal study of risk-glorifying video games and reckless driving. *Psychology of Popular Media Culture*, 1(4):244, 2012.
 - [76] Yoshua Bengio Ian Goodfellow and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.
 - [77] Michael Inzlicht, Brandon J Schmeichel, and C Neil Macrae. Why self-control seems (but may not be) limited. *Trends in cognitive sciences*, 18(3):127–133, 2014.
 - [78] Dietmar Jannach, Paul Resnick, Alexander Tuzhilin, and Markus Zanker. Recommender systems-beyond matrix completion. *Communications of the ACM*, 59(11):94–102, 2016.
 - [79] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Con-

- volutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.
- [80] T. Joachims and A. Swaminathan. Tutorial on counterfactual evaluation and learning for search, recommendation and ad placement. In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 1199–1201, 2016.
 - [81] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. Unbiased learning-to-rank with biased feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 781–789. ACM, 2017.
 - [82] Jie Kang, Kyle Condiff, Shuo Chang, Joseph A Konstan, Loren Terveen, and F Maxwell Harper. Understanding how people use natural language to ask for recommendations. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 229–237. ACM, 2017.
 - [83] Yoshiyuki Kawano and Keiji Yanai. Food image recognition with deep convolutional features. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pages 589–593. ACM, 2014.
 - [84] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
 - [85] Keigo Kitamura, Toshihiko Yamasaki, and Kiyoharu Aizawa. Foodlog: capture, analysis and retrieval of personal food images via web. In *Proceedings of the ACM multimedia 2009 workshop on Multimedia for cooking and eating activities*, pages 23–30. ACM, 2009.
 - [86] Robert C Klesges, Linda H Eck, and JoAnne W Ray. Who underreports dietary intake in a dietary recall? evidence from the second national health and nutrition examination survey. *Journal of consulting and clinical psychology*, 63(3):438, 1995.
 - [87] Bart P Knijnenburg, Saadhika Sivakumar, and Daricia Wilkinson. Recommender systems for self-actualization. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 11–14. ACM, 2016.
 - [88] Ron Kohavi, Roger Longbotham, Dan Sommerfield, and Randal M Henne. Controlled experiments on the web: survey and practical guide. *Data mining and knowledge discovery*, 18(1):140–181, 2009.

- [89] Yehuda Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97, 2010.
- [90] Yehuda Koren, Robert Bell, Chris Volinsky, et al. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.
- [91] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [92] Mounia Lalmas and Liangjie Hong. Tutorial on metrics of user engagement: Applications to news, search and e-commerce. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 781–782. ACM, 2018.
- [93] Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents. In *ICML*, volume 14, pages 1188–1196, 2014.
- [94] Angela R Lebbon and Dene T Hurley. The effects of workplace leisure behavior on work-related behavior. *Journal of Behavioral Studies in Business*, 6:1, 2013.
- [95] Uichin Lee, Zhenyu Liu, and Junghoo Cho. Automatic identification of user goals in web search. In *Proceedings of the 14th international conference on World Wide Web*, pages 391–400. ACM, 2005.
- [96] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 297–306. ACM, 2011.
- [97] Wei Li, Justin Matejka, Tovi Grossman, Joseph A Konstan, and George Fitzmaurice. Design and evaluation of a command recommendation system for software applications. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 18(2):6, 2011.
- [98] Daryl Lim, Julian McAuley, and Gert Lanckriet. Top-n recommendation with missing implicit feedback. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 309–312. ACM, 2015.
- [99] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommen-

- dations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.
- [100] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.
 - [101] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.
 - [102] Leonard A Marascuilo and Joel R Levin. Appropriate post hoc comparisons for interaction and nested hypotheses in analysis of variance designs: The elimination of type iv errors. *American Educational Research Journal*, 7(3):397–421, 1970.
 - [103] Benjamin M Marlin and Richard S Zemel. Collaborative prediction and ranking with non-random missing data. In *Proceedings of the third ACM conference on Recommender systems*, pages 5–12. ACM, 2009.
 - [104] Justin Matejka, Wei Li, Tovi Grossman, and George Fitzmaurice. Communitycommands: command recommendations for software applications. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pages 193–202. ACM, 2009.
 - [105] Julian McAuley, Rahul Pandey, and Jure Leskovec. Inferring networks of substitutable and complementary products. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2015.
 - [106] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–52. ACM, 2015.
 - [107] Austin Meyers, Nick Johnston, Vivek Rathod, Anoop Korattikara, Alex Gorban, Nathan Silberman, Sergio Guadarrama, George Papandreou, Jonathan Huang, and Kevin P Murphy. Im2calories: towards an automated mobile vision food diary. In *ICCV*, pages 1233–1241, 2015.
 - [108] T Mikolov and J Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 2013.

- [109] Katherine L Milkman, Todd Rogers, and Max H Bazerman. Harnessing our inner angels and demons: What we have learned about want/should conflicts and how that knowledge can help us reduce short-sighted decision making. *Perspectives on Psychological Science*, 3(4):324–338, 2008.
- [110] Junta Mizuno, Jun Ogata, and Masataka Goto. A similar content retrieval method for podcast episodes. In *Spoken Language Technology Workshop, 2008. SLT 2008. IEEE*. IEEE, 2008.
- [111] Kher Hui Ng, Victoria Shipp, Richard Mortier, Steve Benford, Martin Flintham, and Tom Rodden. Understanding food consumption lifecycles using wearable cameras. *Personal and Ubiquitous Computing*, 19(7):1183–1195, 2015.
- [112] Tien T Nguyen, Pik-Mai Hui, F Maxwell Harper, Loren Terveen, and Joseph A Konstan. Exploring the filter bubble: the effect of using recommender systems on content diversity. In *Proceedings of the 23rd international conference on World wide web*, pages 677–686. ACM, 2014.
- [113] Jon Noronha, Eric Hysen, Haoqi Zhang, and Krzysztof Z Gajos. Platemate: crowdsourcing nutritional analysis from food photographs. In *UIST*, pages 1–12. ACM, 2011.
- [114] Nutrino. Nutrino. <http://nutrino.co/>, 2016.
- [115] Jun Ogata and Masataka Goto. Podcastle: Collaborative training of acoustic models on the basis of wisdom of crowds for podcast transcription. In *Tenth Annual Conference of the International Speech Communication Association*, 2009.
- [116] Eli Pariser. *The filter bubble: How the new personalized web is changing what we read and how we think*. Penguin, 2011.
- [117] Seung-Taek Park and Wei Chu. Pairwise preference regression for cold-start recommendation. In *Proceedings of the third ACM conference on Recommender systems*, pages 21–28. ACM, 2009.
- [118] Slav Petrov et al. Syntaxnet. <https://github.com/tensorflow/models/tree/master/syntaxnet>, 2016.
- [119] István Pilászy and Domonkos Tikk. Recommending new movies: even

- a few ratings are more valuable than metadata. In *Proceedings of the third ACM conference on Recommender systems*, pages 93–100. ACM, 2009.
- [120] PlateJoy. Custom meal plans & meal planning recipes — platejoy. <https://www.platejoy.com/>, 2016.
 - [121] Al Mamunur Rashid, Istvan Albert, Dan Cosley, Shyong K Lam, Sean M McNee, Joseph A Konstan, and John Riedl. Getting to know you: learning new user preferences in recommender systems. In *ACM IUI*, 2002.
 - [122] Ali Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 806–813, 2014.
 - [123] Steffen Rendle and Christoph Freudenthaler. Improving pairwise learning for item recommendation from implicit feedback. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 273–282. ACM, 2014.
 - [124] 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.
 - [125] Edison Research. The podcast consumer 2017, 2017.
 - [126] William W Ronan, Gary P Latham, and SB Kinne. Effects of goal setting and supervision on worker behavior in an industrial situation. *journal of Applied Psychology*, 58(3):302, 1973.
 - [127] Daniel E Rose and Danny Levinson. Understanding user goals in web search. In *Proceedings of the 13th international conference on World Wide Web*, pages 13–19. ACM, 2004.
 - [128] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.
 - [129] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *NIPS*, 2007.

- [130] Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260. ACM, 2002.
- [131] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. Recommendations as treatments: Debiasing learning and evaluation. In *International Conference on Machine Learning*, pages 1670–1679, 2016.
- [132] Patrick Shafto and Olfa Nasraoui. Human-recommender systems: From benchmark data to benchmark cognitive models. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 127–130. ACM, 2016.
- [133] Amit Sharma, Jake M Hofman, and Duncan J Watts. Estimating the causal impact of recommendation systems from observational data. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, pages 453–470. ACM, 2015.
- [134] Yue Shi, Martha Larson, and Alan Hanjalic. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys (CSUR)*, 47(1):3, 2014.
- [135] Shopwell. Innit - your food. simplified & solved. <https://www.innit.com/shopwell/>, 2016.
- [136] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.
- [137] Apache Spark. Mllib. <https://spark.apache.org/mllib/>, 2017.
- [138] Harald Steck. Training and testing of recommender systems on data missing not at random. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 713–722. ACM, 2010.
- [139] Harald Steck. Item popularity and recommendation accuracy. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 125–132. ACM, 2011.
- [140] Harald Steck. Evaluation of recommendations: rating-prediction and

- ranking. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 213–220. ACM, 2013.
- [141] Ana-Andreea Stoica, Christopher Riederer, and Augustin Chaintreau. Algorithmic glass ceiling in social networks: The effects of social recommendations on network diversity. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 923–932. International World Wide Web Conferences Steering Committee, 2018.
 - [142] Jessica Su, Aneesh Sharma, and Sharad Goel. The effect of recommendations on network structure. In *Proceedings of the 25th international conference on World Wide Web*, pages 1157–1167. International World Wide Web Conferences Steering Committee, 2016.
 - [143] Kyoko Sudo, Kazuhiko Murasaki, Jun Shimamura, and Yukinobu Taniguchi. Estimating nutritional value from food images based on semantic segmentation. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pages 571–576. ACM, 2014.
 - [144] Mingxuan Sun, Fuxin Li, Joonseok Lee, Ke Zhou, Guy Lebanon, and Hongyuan Zha. Learning multiple-question decision trees for cold-start recommendation. In *WSDM*, 2013.
 - [145] Martin Svensson, Kristina Höök, and Rickard Cöster. Designing and evaluating kalas: A social navigation system for food recipes. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 12(3):374–400, 2005.
 - [146] Adith Swaminathan and Thorsten Joachims. Counterfactual risk minimization: Learning from logged bandit feedback. In *International Conference on Machine Learning*, pages 814–823, 2015.
 - [147] Adith Swaminathan and Thorsten Joachims. The self-normalized estimator for counterfactual learning. In *Advances in Neural Information Processing Systems*, pages 3231–3239, 2015.
 - [148] Adith Swaminathan, Akshay Krishnamurthy, Alekh Agarwal, Miro Dudik, John Langford, Damien Jose, and Imed Zitouni. Off-policy evaluation for slate recommendation. In *Advances in Neural Information Processing Systems*, pages 3635–3645, 2017.
 - [149] Liang Tang, Bee-Chung Chen, Deepak Agarwal, and Bo Long. An empirical study on recommendation with multiple types of feedback. In

Proceedings of the 22th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2016.

- [150] Jaime Teevan, Susan T Dumais, and Daniel J Liebling. To personalize or not to personalize: modeling queries with variation in user intent. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 163–170. ACM, 2008.
- [151] Choon Hui Teo, Houssam Nassif, Daniel Hill, Sriram Srinivasan, Mitchell Goodman, Vijai Mohan, and SVN Vishwanathan. Adaptive, personalized diversity for visual discovery. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 35–38. ACM, 2016.
- [152] Edison Thomaz, Aman Parnami, Irfan Essa, and Gregory D Abowd. Feasibility of identifying eating moments from first-person images leveraging human computation. In *Proceedings of the 4th International SenseCam & Pervasive Imaging Conference*, pages 26–33. ACM, 2013.
- [153] Sabina Tomkins, Steven Isley, Ben London, and Lise Getoor. Sustainability at scale: towards bridging the intention-behavior gap with sustainable recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 214–218. ACM, 2018.
- [154] Christoph Trattner and David Elswailer. Investigating the healthiness of internet-sourced recipes: implications for meal planning and recommender systems. In *Proceedings of the 26th international conference on world wide web*, pages 489–498. International World Wide Web Conferences Steering Committee, 2017.
- [155] Manos Tsagkias, Martha Larson, and Maarten De Rijke. Predicting podcast preference: An analysis framework and its application. *Journal of the American Society for information Science and Technology*, 61(2):374–391, 2010.
- [156] Gabrielle M Turner-McGrievy, Elina E Helander, Kirsikka Kaipainen, Jose Maria Perez-Macias, and Ilkka Korhonen. The use of crowdsourcing for dietary self-monitoring: crowdsourced ratings of food pictures are comparable to ratings by trained observers. *Journal of the American Medical Informatics Association*, 22(e1):e112–e119, 2015.
- [157] Mayumi Ueda, Syungo Asanuma, Yusuke Miyawaki, and Shinsuke Nakajima. Recipe recommendation method by considering the user’s preference and ingredient quantity of target recipe. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, 2014.

- [158] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *Advances in neural information processing systems*, pages 2643–2651, 2013.
- [159] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.
- [160] Youri van Pinxteren, Gijs Geleijnse, and Paul Kamsteeg. Deriving a recipe similarity measure for recommending healthful meals. In *Proceedings of the 16th international conference on Intelligent user interfaces*, pages 105–114. ACM, 2011.
- [161] Chong Wang and David M Blei. Collaborative topic modeling for recommending scientific articles. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 448–456. ACM, 2011.
- [162] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1235–1244. ACM, 2015.
- [163] Jason Weston, Samy Bengio, and Nicolas Usunier. Large scale image annotation: learning to rank with joint word-image embeddings. *Machine learning*, 81(1):21–35, 2010.
- [164] Jason Weston, Samy Bengio, and Nicolas Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, volume 11, pages 2764–2770, 2011.
- [165] Jason Weston, Hector Yee, and Ron J Weiss. Learning to rank recommendations with the k-order statistic loss. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 245–248. ACM, 2013.
- [166] Ryen W White and Steven M Drucker. Investigating behavioral variability in web search. In *Proceedings of the 16th international conference on World Wide Web*, pages 21–30. ACM, 2007.
- [167] Jacob O Wobbrock, Leah Findlater, Darren Gergle, and James J Higgins. The aligned rank transform for nonparametric factorial analyses using only anova procedures. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 143–146. ACM, 2011.

- [168] 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.
- [169] Yahoo! *Yahoo! Webscope dataset ydata-ymusic-rating-study-v1*, 2006.
- [170] Ming Yan, Jitao Sang, and Changsheng Xu. Mining cross-network association for youtube video promotion. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 557–566. ACM, 2014.
- [171] Longqi Yang, Eugene Bagdasaryan, Joshua Gruenstein, Cheng-Kang Hsieh, and Deborah Estrin. Openrec: A modular framework for extensible and adaptable recommendation algorithms. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 664–672. ACM, 2018.
- [172] Longqi Yang, Yin Cui, Yuan Xuan, Chenyang Wang, Serge Belongie, and Deborah Estrin. Unbiased offline recommender evaluation for missing-not-at-random implicit feedback. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 279–287. ACM, 2018.
- [173] Longqi Yang, Yin Cui, Fan Zhang, John P Pollak, Serge Belongie, and Deborah Estrin. Plateclick: Bootstrapping food preferences through an adaptive visual interface. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management*, pages 183–192. ACM, 2015.
- [174] Longqi Yang, Chen Fang, Hailin Jin, Matthew D Hoffman, and Deborah Estrin. Personalizing software and web services by integrating unstructured application usage traces. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 485–493. International World Wide Web Conferences Steering Committee, 2017.
- [175] Longqi Yang, Chen Fang, Hailin Jin, Matthew D Hoffman, and Deborah Estrin. Characterizing user skills from application usage traces with hierarchical attention recurrent networks. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 9(6):68, 2018.
- [176] Longqi Yang, Cheng-Kang Hsieh, and Deborah Estrin. Beyond classification: Latent user interests profiling from visual contents analysis. In *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, pages 1410–1416. IEEE, 2015.

- [177] Longqi Yang, Cheng-Kang Hsieh, Hongjian Yang, John P Pollak, Nicola Dell, Serge Belongie, Curtis Cole, and Deborah Estrin. Yum-me: a personalized nutrient-based meal recommender system. *ACM Transactions on Information Systems (TOIS)*, 36(1):7, 2017.
- [178] Longqi Yang, Michael Sobolev, Christina Tsangouri, and Deborah Estrin. Understanding user interactions with podcast recommendations delivered via voice. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 190–194. ACM, 2018.
- [179] Longqi Yang, Michael Sobolev, Yu Wang, Jenny Chen, Drew Dunne, Christina Tsangouri, Nicola Dell, Mor Naaman, and Deborah Estrin. How intention informed recommendations modulate choices: A field study of spokenword content. In *Proceedings of the 2019 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2019.
- [180] Longqi Yang, Yu Wang, Drew Dunne, Michael Sobolev, Mor Naaman, and Deborah Estrin. More than just words: Modeling non-textual characteristics of podcasts. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 276–284. ACM, 2019.
- [181] Mike Yeomans, Anuj Shah, Sendhil Mullainathan, and Jon Kleinberg. Making sense of recommendations. *Preprint at http://scholar.harvard.edu/files/sendhil/files/recommenders55_01.pdf*, 2016.
- [182] Yummly. Yummly — recipe api & food api. <https://developer.yummly.com/>, 2016.
- [183] Fuzheng Zhang, Nicholas Jing Yuan, Kai Zheng, Defu Lian, Xing Xie, and Yong Rui. Exploiting dining preference for restaurant recommendation. In *Proceedings of the 25th International Conference on World Wide Web*, pages 725–735. International World Wide Web Conferences Steering Committee, 2016.
- [184] Xi Zhang, Jian Cheng, Shuang Qiu, Guibo Zhu, and Hanqing Lu. Dualds: A dual discriminative rating elicitation framework for cold start recommendation. *Knowledge-Based Systems*, 2015.
- [185] Xiaoying Zhang, Junzhou Zhao, and John Lui. Modeling the assimilation-contrast effects in online product rating systems: Debiasing and recommendations. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 98–106. ACM, 2017.

- [186] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *Computer Vision–ECCV 2014*, pages 94–108. Springer, 2014.
- [187] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. *NIPS*, 2004.
- [188] Ke Zhou, Shuang-Hong Yang, and Hongyuan Zha. Functional matrix factorizations for cold-start recommendation. In *SIGIR*, 2011.
- [189] Zipongo. Personalizing food recommendations with data science. <http://blog.zipongo.com/blog/2015/8/11/personalizing-food-recommendations-with-data-science>, 2015.
- [190] Zipongo. Corporate nutrition programs - manage nutrition with zipongo. <https://meetzipongo.com/>, 2016.