Structural Analysis of User Choices for Mobile App Recommendation

BIN LIU, IBM Thomas J. Watson Research Center
YAO WU, Simon Fraser University
NEIL ZHENQIANG GONG, Iowa State University
JUNJIE WU, Beihang University
HUI XIONG, Rutgers University
MARTIN ESTER, Simon Fraser University

Advances in smartphone technology have promoted the rapid development of mobile apps. However, the availability of a huge number of mobile apps in application stores has imposed the challenge of finding the right apps to meet the user needs. Indeed, there is a critical demand for personalized app recommendations. Along this line, there are opportunities and challenges posed by two unique characteristics of mobile apps. First, app markets have organized apps in a hierarchical taxonomy. Second, apps with similar functionalities are competing with each other. Although there are a variety of approaches for mobile app recommendations, these approaches do not have a focus on dealing with these opportunities and challenges. To this end, in this article, we provide a systematic study for addressing these challenges. Specifically, we develop a *structural user choice model* (SUCM) to learn fine-grained user preferences by exploiting the hierarchical taxonomy of apps as well as the competitive relationships among apps. Moreover, we design an efficient learning algorithm to estimate the parameters for the SUCM model. Finally, we perform extensive experiments on a large app adoption dataset collected from Google Play. The results show that SUCM consistently outperforms state-of-the-art Top-N recommendation methods by a significant margin.

CCS Concepts: • **Information systems → Data mining**; **Recommender systems**; **Electronic commerce;**

Additional Key Words and Phrases: Recommender systems, mobile apps, hierarchy structure, structural choices

This article was supported in part by the Natural Science Foundation of China (71329201) and the Rutgers 2015 Chancellor's Seed Grant Program. Y. Wu and M. Ester were supported in part by a Discovery Grant from the National Science and Engineering Research Council of Canada (250960-2012). N. Z. Gong was supported by the Department of Electrical and Computer Engineering, and College of Engineering at Iowa State University. J. Wu was supported in part by the National Natural Science Foundation of China (71322104, 71531001, 71471009, 71490723, 71171007), the National High Technology Research and Development Program of China (SS2014AA012303), and the Fundamental Research Funds for the Central Universities.

Authors' addresses: B. Liu, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598; email: BenBinLiu@gmail.com; Y. Wu and M. Ester, School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada; emails: {wuyaow, ester}@sfu.ca; N. Z. Gong, Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011; email: neilgong@iastate.edu; J. Wu, School of Economics and Management, Beihang University, Beijing 100191, China; email: wujj@buaa.edu.cn; H. Xiong (corresponding author), Department of Management Science and Information Systems, Rutgers Business School, Rutgers University, Newark, NJ 07102; email: hxiong@rutgers.edu.

DOI: http://dx.doi.org/10.1145/2983533

ACM Reference Format:
Bin Liu, Yao Wu, Neil Zhenqiang Gong, Junjie Wu, Hui Xiong, and Martin Ester. 2016. Structural analysis of user choices for mobile app recommendation. ACM Trans. Knowl. Discov. Data 11, 2, Article 17 (November 2016), 23 pages.
DOI: http://dx.doi.org/10.1145/2983533

1. INTRODUCTION

Recent years have witnessed the tremendous growth in mobile devices among an increasing number of users and the penetration of mobile devices into every component of modern life. Indeed, the smartphone market surpassed the PC market in 2011 for the first time in history.[1] Thereafter, the smartphone market has continued to increase dramatically, e.g., the smartphones shipped in the third quarter of 2013 increased 44% year-on-year.[2] One of the reasons lies in the fact that users are able to augment the functions of mobile devices by taking the advantage of various feature-rich third-party *applications* (or apps for brevity), which can be easily obtained from centralized markets such as Google Play and App Store. However, the availability of a huge number of mobile apps in application stores has imposed the challenge of finding the right apps to meet the user needs. For instance, as of July 2013, Google Play had over 1 million apps with over 50 billion cumulative downloads, and the number of apps had reached over 1.4 million in January 2015[3]; as of February 2015, App Store had over 1.4 million apps and a cumulative of over 100 billion apps downloaded.[4] As a result, there is a critical demand for effective personalized app recommendations.

However, for the development of personalized app recommender systems, there are opportunities and challenges posed by two unique characteristics of mobile apps. First, application stores have organized apps in a hierarchical taxonomy. For instance, Google Play groups the apps into 27 categories, such as *social*, *games*, and *sports* according to their functionalities. These categories can be further divided into subcategories, e.g., apps in the category of *games* are further divided into subcategories such as *action*, *arcade*, and *puzzle*. For the apps in the same category or subcategory, they have similar functionalities. Then, how a user navigates through the hierarchy to locate relevant apps represents a fine-grained interest preference of the user. Thus, the first challenge is how to leverage this hierarchical taxonomy of apps to better profile user interests and enhance app recommendations. Second, apps with similar functionalities are competing with each other. For instance, when a user has already adopted Google Maps as his/her navigation tool, the user might not be interested in other navigation tools such as Apple Maps. Although there are a variety of existing approaches for mobile app recommendations, these approaches do not have a focus on dealing with these opportunities and challenges.

Instead, in this article, we provide a systematic study to address these challenges. Specifically, we first develop a *structural user choice model* (SUCM) to learn fine-grained user preferences by exploiting the hierarchical taxonomy of apps as well as the competitive relationships among apps.[5] Since apps are organized as a hierarchical taxonomy, we model the user choice as two phases. In the first phase, a user decides which type of apps to choose and then moves to the appropriate app category/subcategory. In the

[1]The Smartphone Market is Bigger Than the PC Market (2011), http://www.businessinsider.com/smartphone-bigger-than-pc-market-2011-2.
[2]Smartphone Sales in the Third Quarter of 2013 (2013), http://www.finfacts.ie/irishfinancenews/article_1026800.shtml.
[3]Google Play Statistics, Retrieved January 2015, http://en.wikipedia.org/wiki/Google_Play.
[4]App Store Statistics, Retrieved January 2015, http://en.wikipedia.org/wiki/App_Store_(iOS).
[5]Note that the model and algorithms developed in this article can also be applied to other domains in which items are organized in a hierarchical way.

second phase, the user chooses apps in the selected category/subcategory. Such structural user choice is modeled by a unique *choice path* over the tree hierarchy, wherein the choice path starts from the root of the hierarchy and goes down to the app that is selected by a user. In each step of moving along the choice path, the competitions between the candidates (i.e., either the same level categories/subcategories or apps in a chosen category/subcategory) play an important role in affecting user's choices. We capture the structural choice procedure by cascading user preferences over the choice paths through a probabilistic model. Specifically, in our probabilistic model, motivated by the widely used discrete choice models in economics [Luce 1959; McFadden 1973; Manski 1977], we model the probability that a user reaches a certain node in the choice path as a *softmax* of the user's preference on the chosen node over the user's preference on all the nodes at the sample level. The softmax function is used to capture the competitions between categories/subcategories or apps in a category/subcategory. Moreover, we model a user's preference over one node using latent factors, which enables us to capture the correlations between nodes.

Moreover, we design an efficient learning algorithm to estimate the parameters of the SUCM model. The major challenge of learning the parameters lies in the softmax on the leaf nodes (apps) of the tree hierarchy. Indeed, it is impractical to optimize these softmax functions for a subcategory of apps by directly applying Stochastic Gradient Descent (SGD), because the time complexity of one SGD step is linear to the number of apps under the subcategory, which might be very large. To address this challenge, we relax the softmax term in each subcategory into a hierarchical softmax; thus, the time complexity of learning parameters is reduced to be logarithm of the number of apps under the subcategory.

Finally, we collected a large-scale dataset from Google Play to evaluate our approach and compare SUCM with state-of-the-art approaches. The experimental results show that SUCM consistently outperforms these methods with a significant margin in terms of a variety of widely used evaluation metrics for Top-N recommendation.

2. PROBLEM DEFINITION

We first introduce three key concepts and then formally define our app recommendation problem.

Definition 2.1 (*Category Tree*). A Category Tree (denoted as Γ) is a data structure to organize apps according to their properties (e.g., functionalities). Figure 1 shows an example Category Tree adopted by Google Play. In a Category Tree, internal nodes represent categories or subcategories, leaf nodes represent apps, and the children of an internal node represent the subcategories or the apps that belong to the category/subcategory represented by the node. We use z to denote an internal node in Γ, and we denote by $\text{level}(z)$, $c(z)$, $\pi(z)$, and $s(z)$ the level, the children, the parents, and the siblings of z, respectively. Moreover, we use z_M to denote an internal node whose children are leaf nodes and use i to represent an app.

We note that an app might belong to multiple categories due to the rich functionalities provided by it, which makes the category hierarchy not a tree. However, we found that mobile markets such as Google Play do not place an app into multiple categories based on the dataset we collected from Google Play, and thus we do not consider this scenario.

Definition 2.2 (*Choice Path*). A choice path is a sequence of nodes that a user traverses through the Category Tree Γ, starting from the root and ending at a leaf node that corresponds to the app selected by the user. For instance, if a user adopts an app i, the choice path can be represented as $\text{path}_i = z_0 \rightarrow z_1 \rightarrow \cdots \rightarrow z_M \rightarrow i$. Note that, given the Category Tree Γ, the choice path path_i for app i is unique.

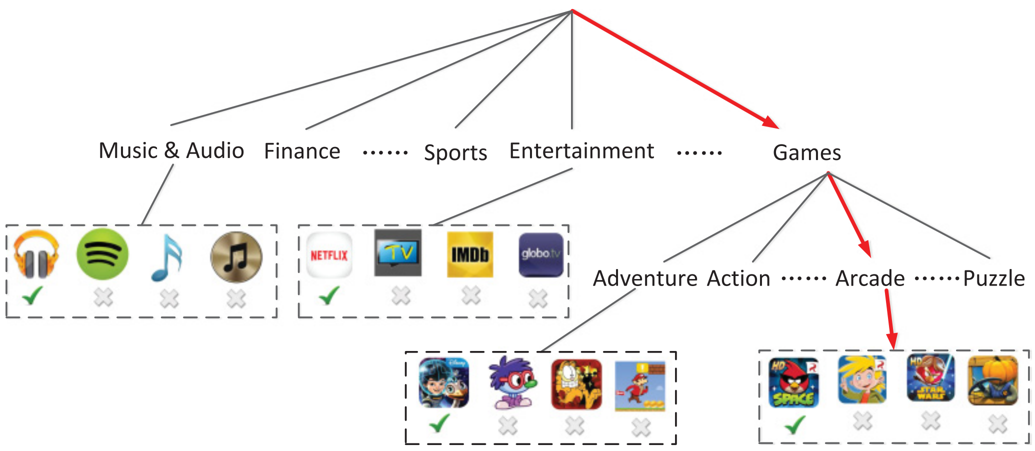


Fig. 1. An illustrative example of structural user choice for app adoption in Google Play. First, apps are organized into a Category Tree. Second, as illustrated by the highlighted and arrowed path, a user makes an app adoption by traversing a *choice path* from the root of the tree to the chosen app.

Definition 2.3 (*Competing Apps*). For an app i, we denote by $\mathcal{A}(i)$ the set of apps that have competing properties (e.g., functionalities) and compete with i to attract users. In this article, we treat the siblings of an app i under the same category/subcategory in the Category Tree as the competing apps.

We note that users might have multiple ways to adopt apps, e.g., suggestions from friends, recommendations from Google Play store, and so forth. However, we assume that no matter in which way a user is aware of an app, the decision is made on the functionality of the app and its competitors with similar functionalities, thus following the choice path we discuss above.

It also should be noted that we do not assume a user only adopts one app in a subcategory. The category/subcategory in the Category Tree provided by mobile markets such as Google Play is not fine grained enough so some siblings of the app i might provide slightly different functionalities with i. For example, Facebook, LinkedIn, and Twitter all belong to *Social* category. We model the process of one user adopting an app using a structural choice model. If a user selects multiple apps under a same category, the joint probability of selecting them together would be optimized (see details in Section 3).

Given the above three concepts, we can formally define our app recommendation problem as follows: Suppose we are given a set of users denoted as $\mathcal{U} = \{1, 2, \ldots, U\}$, a set of apps denoted as $\mathcal{I} = \{1, 2, \ldots, I\}$; the apps are organized into a predefined Category Tree Γ, each app i has a set of competing apps $\mathcal{A}(i)$, a set of adoption records $\{(u, i)\}$ indicating which users have adopted which apps, and then our goal is to recommend each user a list of apps that matches his/her interest preference. In the rest of the article, we use u to index users, and i and j to index apps. Moreover, we use the two terms *app* and *item* interchangeably. Table I shows some important notations used in this article.

3. STRUCTURAL USER CHOICE MODEL

In this section, we present our SUCM to learn fine-grained user interest preference via leveraging the Category Tree and competitions between apps for app recommendation.

Table I. Mathematical Notations

Symbol	Description
u	User index for user set $\mathcal{U} = \{1, 2, \ldots, U\}$
i, j	App index for app set $\mathcal{I} = \{1, 2, \ldots, I\}$
Γ	Predefined Category Tree
z	Internal node in Category Tree Γ, in particular, z_M denotes a node whose children are leaf nodes
path_i	Choice path in Γ: $z_0 \to z_1 \to \cdots \to z_M \to i$
$\pi(z), s(z), c(z)$	Parent, sibling, and children of internal node z in the Category Tree Γ
$\mathbf{p}_u, \mathbf{q}_i, \mathbf{q}_z$	Latent factor vector for user u, app i, and internal node z in Γ
b_i, b_z	Bias term for app i and internal node z
y_{ui}	Affinity score of user u for app i
y_{uz}	Affinity score of user u for internal node z
$\mathcal{D} = \{(u, i)\}$	Observed user–app adoption instances
$\mathcal{D}_u$	Adopted apps by user u

3.1. Model Structural User Choice

As shown in Figure 1, given a Category Tree Γ, there exists one unique *choice path* from the root node to app i, namely,

$$\text{path}_i = \underbrace{z_0 \to z_1 \to \cdots \to z_M}_{\substack{\text{Phase I:}\\ \text{locate a subcategory}}} \underbrace{\longrightarrow i.}_{\substack{\text{Phase II:}\\ \text{choose an app}}}$$

We see that the structural user choice consists of two adoption phases. In the first phase, a user decides what types of apps to choose and moves to the appropriate category or subcategory in the Category Tree, namely, traverses $z_0 \to z_1 \to \cdots \to z_M$. In the second phase, the user makes app adoption decisions by choosing app i among all competing apps under the located subcategory z_M. For example, if a user wants to select the app *Angry Birds* under the subcategory *Arcade*, he would first consider the *Games* category and then further locates himself at the *Arcade* subcategory before he finally chooses app *Angry Birds*.

We model the process of a user u traversing path $z_0 \to z_1 \to \cdots \to z_M \to i$ as a sequence of decisions made for the multiple competing choices at each choice step. Specifically, in each step among this decision-making sequence:

—for choosing category or subcategory, user u chooses one child node z from all the children $c(\pi(z))$ of z's parent node $\pi(z)$;
—for choosing app, user u chooses app i from all the children of i's parent node, namely z_M.

Each decision-making step can be seen as a discrete choice model, whose theoretical foundation is the neoclassical economic theory on preferences and utility built on a set of axiomatic assumptions [Luce 1959; McFadden 1973; Manski 1977]. The discrete choice model implies that a user u is endowed a *utility value* $f(u, z)$ to each alternative z in a choice set $\mathcal{A}(z)$. In our recommendation task, the utility value $f(u, z)$ can be the affinity score, which captures user preferences, between user u and choice z. Following the random utility model [Manski 1977], we model the utility as a random variable:

$$\nu_{uz} = f(u, z) + \varepsilon_{uz}, \tag{1}$$

where $f(u, z)$ is the deterministic part of the utility reflecting user preference, and ε_{uz} is the stochastic part capturing the impact of all unobserved factors that affect the user's choice. By assuming the stochastic part ε_{uz} to be an independently and identically distributed log Weibull (type I extreme value) distribution, we can obtain the multinomial choice model [McFadden 1973]. Specifically, in a multinomial choice model, the probability of a user u choosing z from a choice set $\mathcal{A}(z)$ takes the form of

$$\Pr(\text{user } u \text{ choose } z|\mathcal{A}(z)) = \frac{\exp(f(u,z))}{\sum_{z' \in \mathcal{A}(z)} \exp(f(u,z'))}, \tag{2}$$

where $f(u,z)$ is a user-preference-dependent utility function. This choice model also holds for user u choosing app i from app choice set $\mathcal{A}(i)$. Note that the choice model $\frac{\exp(f(u,z))}{\sum_{z' \in \mathcal{A}(z)} \exp(f(u,z'))}$ turns out to be a *softmax* function of utility value $f(u,z)$. In the following, we elaborate how we model each phase.

Phase I: Model category/subcategory preference. Following the latent factor models that are widely used in conventional recommender systems [Salakhutdinov and Mnih 2008; Koren 2008], we use a latent factor vector $\mathbf{p}_u \in \mathbb{R}^K$ to represent a user's latent interest, where K is the dimension of the latent factor vector. Intuitively, $\mathbf{p}_u$ captures the interest of the user u. To capture the hierarchical structural user choice, we associate an internal node z in the Category Tree with a latent factor vector $\mathbf{q}_z$, which represents the properties (e.g., functionalities) of z in the latent space. Moreover, we define the affinity score between a user u and an internal node z as

$$y_{uz} = b_z + \mathbf{p}_u^\top \mathbf{q}_z, \tag{3}$$

where b_z is a bias term for the node z. The category/subcategory node affinity score represents the preference of a user over the category or the subcategory of apps (e.g., Games).

We model the process of a user locating a subcategory as a sequence of decisions made for the multiple competing choices, starting from the root node and moving along the Category Tree toward the internal node corresponding to the subcategory. Specifically, in each step among this decision-making sequence, user u chooses one child node z from all the children of z's parent node $\pi(z)$. Following the choice model as shown in Equation (2), we assume the utility as the affinity score between user u and internal node z, i.e.,

$$f(u,z) = y_{uz} = b_z + \mathbf{p}_u^\top \mathbf{q}_z.$$

Then, we model the probability of user u choosing the child z from all the children $c(\pi(z))$ of z's parent node $\pi(z)$ as a *softmax* function of the affinity scores between the user u and the internal nodes $c(\pi(z))$. Formally, we have

$$\Pr(z|u,\pi(z)) = \frac{\exp(y_{uz})}{\sum_{z' \in c(\pi(z))} \exp(y_{uz'})}. \tag{4}$$

The *softmax* function is used to model the competitions between the nodes in $c(\pi(z))$. As a result, the probability of user u traverses $z_0 \to z_1 \to \cdots \to z_M$ to reach the subcategory z_M is cascaded as

$$\begin{aligned} \Pr(z_0 \to z_1 \to \cdots \to z_M|u) &= \prod_{m=1}^{M} \Pr(z_m|u, z_{m-1}) \\ &= \prod_{m=1}^{M} \frac{\exp(y_{uz})}{\sum_{z' \in c(z_{m-1})} \exp(y_{uz'})} \\ &= \prod_{m=1}^{M} \frac{\exp\left(b_z + \mathbf{p}_u^\top \mathbf{q}_z\right)}{\sum_{z' \in c(z_{m-1})} \exp\left(b_{z'} + \mathbf{p}_u^\top \mathbf{q}_{z'}\right)}. \end{aligned} \tag{5}$$

Phase II: Model app adoption. After a user locates at a specific subcategory node z_M whose children are all apps, the user makes an app adoption decision by choosing an app

i among all competing choices $c(z_M)$. We use a latent factor vector $\mathbf{q}_i \in \mathbb{R}^K$ to represent the latent factor of app i. Intuitively, $\mathbf{q}_i$ encodes the properties (e.g., functionalities) of app i. Moreover, we define the affinity score between user u and app i as

$$y_{ui} = b_i + \mathbf{p}_u^\top \mathbf{q}_i, \tag{6}$$

where b_i is a bias term for app i. Again, following the choice model as shown in Equation (2), we assume the utility as the affinity score between user u and app i, i.e.,

$$f(u, i) = y_{ui} = b_i + \mathbf{p}_u^\top \mathbf{q}_i.$$

Then, we model the probability of user u selecting app i over its competing alternatives under the subcategory node z_M using a softmax function as follows:

$$\begin{aligned} \Pr(i|u, z_M) &= \frac{\exp(y_{ui})}{\sum_{j\in c(z_M)} \exp(y_{uj})} \\ &= \frac{\exp\left(b_i + \mathbf{p}_u^\top \mathbf{q}_i\right)}{\sum_{j\in c(z_M)} \exp\left(b_j + \mathbf{p}_u^\top \mathbf{q}_j\right)}, \end{aligned} \tag{7}$$

where z_M is the parent node of app i and $c(z_M)$ includes all competing apps of app i and i itself. The softmax function is used to model the competitions between apps.

Model the overall structural choice probability. Note that there exists one unique *choice path* from the root node to app i, namely,

$$\text{path}_i = z_0 \to z_1 \to \cdots \to z_M \to i.$$

Then, the probability of user u choosing app i is the joint probability of u selecting each node in the choice path path_i, i.e., we have

$$\begin{aligned} \Pr(i|u) &= \Pr(i|u, z_M) \times \Pr(z_0 \to z_1 \to \cdots \to z_M|u) \\ &= \Pr(i|u, z_M) \prod_{m=1}^{M} \Pr(z_m|u, z_{m-1}) \\ &= \frac{\exp\left(b_i + \mathbf{p}_u^\top \mathbf{q}_i\right)}{\sum_{j\in c(z_M)} \exp\left(b_j + \mathbf{p}_u^\top \mathbf{q}_j\right)} \prod_{m=1}^{M} \frac{\exp\left(b_{z_m} + \mathbf{p}_u^\top \mathbf{q}_{z_m}\right)}{\sum_{z'\in c(z_{m-1})} \exp\left(b_{z'} + \mathbf{p}_u^\top \mathbf{q}_{z'}\right)}, \end{aligned} \tag{8}$$

where the first term $\Pr(i|u, z_M)$ is user u's adoption probability of app i under subcategory node z_M and the second term $\prod_{m=1}^{M} \Pr(z_m|u, z_{m-1})$ captures the structural choice by cascading user preferences over the Category Tree Γ.

3.2. Model Structural App Dependences

Intuitively, nodes that are closer in the Category Tree Γ could have more similar properties. For instance, apps under the subcategory *action* are more similar to those under the subcategory *arcade* than those under the category *weather* because both *action* and *arcade* belong to the *games* category. Thus, we associate each internal node z with a latent variable $\mathbf{q}_z$ to represent the category/subcategory level properties, and we model the latent variable $\mathbf{q}_z$ as a function of the latent variable of z's parent node $\mathbf{q}_{\pi(z)}$ to capture the hierarchical structural dependences between the nodes in the Category Tree. Formally, we have

$$\mathbf{q}_z \sim \begin{cases} \mathcal{N}(0, \sigma^2\mathbf{I}) & \text{if } z \text{ is the root node} \\ \mathcal{N}(\mathbf{q}_{\pi(z)}, \sigma^2\mathbf{I}) & \text{otherwise,} \end{cases} \tag{9}$$

where $\mathcal{N}(u, \sigma^2)$ is a normal distribution with mean u and standard deviation σ.

3.3. Discussion

Note that our model does not only capture the competitions between apps under the same categories, but also incorporates the correlations between apps via the latent factor representations.

—*Competition*. We use a softmax function to model the probability of selecting a child node (a subcategory or an app) under a category node. If user u selects a child node z from all the competing nodes $\mathcal{A}(z)$, the value of y_{uz} should be larger than all other $y_{uz'}$ where $z' \in \mathcal{A}(z)$ and $z \neq z'$. This model characteristic can address the cases when multiple apps in same categories are adopted.

—*Correlation*. The latent factor model is able to model the correlations between apps and categories. For example, if two categories are always liked by the same users, the latent factors of them will be close to each other in the latent space. As a result, if we know a user likes one of the two categories, the value of his/her preferences on the other one will also be large.

Here, we highlight some important differences between our model and previous work (see Section 5.2 and Section 5.4 for details):

—Instead of fitting a point-wise regression model (e.g., PMF [Salakhutdinov and Mnih 2008] and LLFM [Agarwal and Chen 2009]), the proposed model SUCM optimizes the choice decision making through choice probabilities cascaded in the hierarchy structure.

—Previous feature-based latent factor approaches (e.g., SVDFeature [Chen et al. 2012] and LibFM [Rendle 2010, 2012]) utilize item features by representing the user preference on an item using a linear combination of the user–item affinities and the user–feature affinities. Differently, SUCM is designed for structurally organized features, and it models the structure by cascading the choice probabilities instead of linear combinations.

—SUCM generalizes the *flat* choice model – Collaborative Competitive Filtering (CCF) [Yang et al. 2011] – to a *structural* choice model via leveraging the hierarchy information. We also present an efficient learning algorithm based on Hierarchical Softmax (See Section 4.1) that can also be used for CCF.

4. PARAMETER ESTIMATION

Let $\Theta = \{\mathbf{p}_u, \mathbf{q}_i, \mathbf{q}_z, b_i, b_z\}_{u\in\mathcal{U}, i\in\mathcal{I}, z\in\Gamma}$ denote all parameters to be estimated. Given the observed user–app adoption records $\mathcal{D} = \{(u, i, \text{path}_i)\}$ and the category tree Γ, we have the posterior probability distribution of the parameters as follows:

$$\Pr(\Theta|\mathcal{D},\Gamma) \propto \prod_{u=1}^{U}\prod_{i\in\mathcal{D}_u}\Pr(i|u,z_M)\prod_{m=1}^{M}\Pr(z_m|u,z_{m-1})\prod_{\substack{m=1\\ \forall z\in\Gamma}}^{M}\Pr\big(\mathbf{q}_{z_m}|\mathbf{q}_{z_{m-1}},\sigma^2\mathbf{I}\big), \tag{10}$$

where the first term captures the structural user choices and the second term represents the hierarchical structural dependences of the nodes in the category tree. We estimate all the parameters via maximizing the log likelihood of the posterior:

$$\begin{aligned}\arg\max_{\Theta}\Bigg\{&\sum_{u=1}^{U}\sum_{i\in\mathcal{D}_u}\ln\Pr(i|u,z_M) + \sum_{u=1}^{U}\sum_{i\in\mathcal{D}_u}\sum_{m=1}^{M}\ln\Pr(z_m|u,z_{m-1})\\ &+\sum_{\substack{m=1\\ \forall z\in\Gamma}}^{M}\ln\Pr\big(\mathbf{q}_{z_m}|\mathbf{q}_{z_{m-1}},\sigma^2\mathbf{I}\big)\Bigg\}.\end{aligned} \tag{11}$$

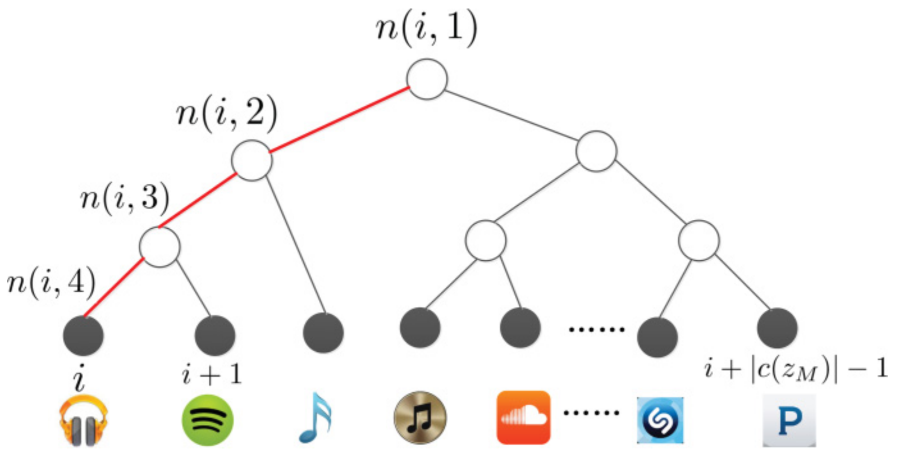


Fig. 2. An illustrative example of binary tree for hierarchical softmax under a category/subcategory. All apps under the category/subcategory (e.g., Music & Audio) are organized using a binary tree. The black nodes (leaf nodes) are apps, and the white nodes are internal nodes. One example path from root node to app i is highlighted as $n(i,1) \rightarrow n(i,2) \rightarrow n(i,3) \rightarrow n(i,4)$, which means the path length $L(i) = 4$.

Note that the widely used regularizations for latent factor vectors [Salakhutdinov and Mnih 2008; Koren et al. 2009] can be applied here, but we exclude the regularization priors for presentation simplicity.

4.1. Hierarchical Softmax

One challenge of directly solving the objective function as shown in Equation (11) rests in the updating of all the parameters over the probability distribution $\Pr(i|u, z_M)$, namely, the first term in Equation (11):

$$\begin{aligned}\sum_{u=1}^{U}\sum_{i\in\mathcal{D}_u}\ln\Pr(i|u,z_M) &= \sum_{u=1}^{U}\sum_{i\in\mathcal{D}_u}\ln\frac{\exp\left(b_i+\mathbf{p}_u^\top\mathbf{q}_i\right)}{\sum_{j\in c(z_M)}\exp\left(b_j+\mathbf{p}_u^\top\mathbf{q}_j\right)}\\ &= \sum_{(u,i)\in\mathcal{D}}\left\{\left(b_i+\mathbf{p}_u^\top\mathbf{q}_i\right)-\ln\left[\sum_{j\in c(z_M)}\exp\left(b_j+\mathbf{p}_u^\top\mathbf{q}_j\right)\right]\right\},\end{aligned}\tag{12}$$

where $c(z_M)$ represents all the apps under the subcategory z_M. The updating computation cost for all the parameters in one user–app adoption instance (u, i) is linear to the number of apps under z_M, which might be very large. To address this challenge, we leverage *hierarchical softmax* to approximate $\Pr(i|u, z_M)$ efficiently. Hierarchical softmax was first introduced by Morin and Bengio [2005] for neural networks and recently was widely used in deep learning [Mikolov et al. 2013a, 2013b]. The main advantage of hierarchical softmax is that, in each training instance, instead of evaluating the parameters for all the children of z_M, we only need to evaluate parameters for $\log|c(z_M)|$ nodes.

Adapting hierarchical softmax to our model is challenging since our hierarchical category tree has multiple layers and applying hierarchical softmax to different layers results in different performances. In our work, since the major computation cost comes from the large number of apps, we adapt hierarchical softmax to the apps. Specifically, we organize the apps under a subcategory using a binary tree. As shown in Figure 2, we represent each app (black nodes) as a leaf node of the binary tree, and the leaf nodes are connected by internal nodes (white nodes). Let $n(i,l)$ be the lth node on the path from the root of the binary tree to i, and let $L(i)$ be the length of this path, and then $n(i,1)$ is the root and $n(i,L(i)) = i$. For each leaf node (i.e., an app), there exists a unique path from the root to the node. Let $n(i,l+1) = \text{left}(n(i,l))$ indicate that $n(i,l+1)$ is the left child node of $n(i,l)$ and we define a sign function as follows:

$$\mathbb{S}(n(i,l+1) = \text{left}(n(i,l))) := \begin{cases} 1 & n(i,l+1) \text{ on left,} \\ -1 & \text{otherwise.} \end{cases} \tag{13}$$

Let $y_{u,n(i,l)}$ be the affinity score between user u and node $n(i,l)$, which is defined as

$$y_{u,n(i,l)} = b_{n(i,l)} + \mathbf{p}_u^\top \mathbf{q}_{n(i,l)}, \tag{14}$$

where $\mathbf{q}_{n(i,l)} \in \mathbb{R}^K$ is the latent factor vector and $b_{n(i,l)}$ is the bias term for node $n(i,l)$. Intuitively, at each inner node $n(i,l)$ in the hierarchical softmax binary tree, we assign the probability of moving left as

$$\Pr(u, n(i,l+1) = \text{left}(n(i,l))) = \sigma\big(b_{n(i,l)} + \mathbf{p}_u^\top \mathbf{q}_{n(i,l)}\big), \tag{15}$$

where $\sigma(x)$ is a sigmoid function defined as follows:

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{16}$$

Accordingly, the probability of moving right is

$$\begin{aligned} \Pr(u, n(i,l+1) \neq \text{left}(n(i,l))) &= 1 - \sigma\big(b_{n(i,l)} + \mathbf{p}_u^\top \mathbf{q}_{n(i,l)}\big) \\ &= \sigma\big(-\big(b_{n(i,l)} + \mathbf{p}_u^\top \mathbf{q}_{n(i,l)}\big)\big). \end{aligned} \tag{17}$$

Combing Equations (15) and (17), we can derive the probability of moving from node $n(i,l)$ to node $n(i,l+1)$ as

$$\Pr(u, n(i,l) \to n(i,l+1)) = \sigma\big(\mathbb{S}(n(i,l+1) = \text{left}(n(i,l))) \cdot \big(b_{n(i,l)} + \mathbf{p}_u^\top \mathbf{q}_{n(i,l)}\big)\big). \tag{18}$$

As a result, by using the path $n(i,1) \to n(i,2) \cdots \to n(i,L(i))$ in the defined hierarchical softmax binary tree, we approximate the probability $\Pr(i|u,z_M)$ as follows:

$$\begin{aligned} \Pr(i|u,z_M) &= \prod_{l=1}^{L(i)-1} \Pr(u, n(i,l) \to n(i,l+1)) \\ &= \prod_{l=1}^{L(i)-1} \sigma\big(\mathbb{S}(n(i,l+1) = \text{left}(n(i,l))) \cdot \big(b_{n(i,l)} + \mathbf{p}_u^\top \mathbf{q}_{n(i,l)}\big)\big). \end{aligned} \tag{19}$$

Note that, instead of computing the affinity scores for all the apps under subcategory z_M to get the probability distribution $\Pr(i|u,z_M)$ as defined in Equation (12), we only need to compute $L(i) - 1$ times in the order of $\log|c(z_M)|$. Also hierarchical softmax does not increase the number of parameters to be estimated. Instead of estimating parameters of $|c(z_M)|$ apps, we only need to estimate the parameters for $|c(z_M)| - 1$ internal nodes.

Comments. The binary tree built for hierarchical softmax is meant for computation efficiency purpose, which is different form the category tree Γ used for structural choice modeling.

4.2. Parameter Learning

After building the hierarchical softmax binary tree for each most outside subcategory node z_M in the category hierarchy, the unique structural path for user u to choose app i is extended as

$$\text{path}_i = z_0 \rightarrow z_1 \cdots \rightarrow z_M \rightarrow n(i,1) \cdots \rightarrow n(i, L(i)).$$

We rewrite the log likelihood $\ell(\Theta)$ and get the following objective function:

$$\begin{aligned}
\mathcal{O} =& \sum_{u=1}^{U} \sum_{i \in \mathcal{D}_u} \sum_{l=1}^{L(i)-1} \ln \Pr(u, n(i,l) \rightarrow n(i, l+1)) \\
&+ \sum_{u=1}^{U} \sum_{i \in \mathcal{D}_u} \sum_{m=1}^{M} \ln \Pr(z_m | u, z_{m-1}) + \sum_{\substack{m=1 \\ \forall z \in \Gamma}}^{M} \ln \Pr(\mathbf{q}_{z_m} | \mathbf{q}_{z_{m-1}}, \sigma^2 \mathbf{I}) \\
=& \sum_{u=1}^{U} \sum_{i \in \mathcal{D}_u} \sum_{l=1}^{L(i)-1} \ln \sigma \left(\mathbb{S}(n(i, l+1) = \text{left}(n(i,l))) \cdot y_{u,n(i,l)} \right) \\
&+ \sum_{u=1}^{U} \sum_{i \in \mathcal{D}_u} \sum_{m=1}^{M} \ln \frac{\exp\left(b_{z_m} + \mathbf{p}_u^\top \mathbf{q}_{z_m}\right)}{\sum_{z' \in c(z_{m-1})} \exp\left(b_{z'} + \mathbf{p}_u^\top \mathbf{q}_{z'}\right)} \\
&+ \sum_{\substack{m=1 \\ \forall z \in \Gamma}}^{M} \ln \mathcal{N}\left(\mathbf{q}_{z_m} | \mathbf{q}_{z_{m-1}}, \sigma^2 \mathbf{I}\right).
\end{aligned}$$

Note that here we have an updated set of parameters to estimate, namely, $\Theta = \{\mathbf{p}_u, \mathbf{q}_z, \mathbf{q}_{n(i,l)}, b_z, b_{n(i,l)}\}$. Instead of estimating $\mathbf{p}_i$ and b_i for $i \in \mathcal{I}$, we estimate that of internal nodes $n(i,l)$ in the hierarchical softmax binary trees.

We use the stochastic gradient ascent method to update the latent factor variables. Stochastic gradient ascent (descent) has been widely used for many machine learning tasks [Bottou 2010]. The main process involves randomly scanning training instances and iteratively updating parameters. In each iteration, we randomly sample a user–app adoption instance $\langle u, i, \text{path}_i \rangle$, and we maximize $\mathcal{O}(\Theta)$ using the following update rule for Θ:

$$\Theta = \Theta + \epsilon \cdot \frac{\partial \mathcal{O}(\Theta)}{\partial \Theta}, \tag{20}$$

where ϵ is a learning rate.

Specifically, given a user–app adoption instance $\langle u, i, \text{path}_i \rangle$, the gradient with respect to $\mathbf{p}_u$ is

$$\begin{aligned}
\frac{\partial \mathcal{O}}{\partial \mathbf{p}_u} &= \sum_{l=1}^{L(i)-1} \frac{\partial \ln \Pr(u, n(i,l) \rightarrow n(i,l+1))}{\partial \mathbf{p}_u} + \sum_{m=1}^{M} \frac{\partial \ln \Pr(z_m | u, z_{m-1})}{\partial \mathbf{p}_u} \\
&= \sum_{l=1}^{L(i)-1} \left(\mathbf{1}_{l+1} - \sigma\left(y_{u,n(i,l)}\right)\right) \cdot \mathbf{q}_{n(i,l)} \\
&+ \sum_{m=1}^{M} \left(\mathbf{q}_{z_m} - \frac{\sum_{z' \in c(z_{m-1})} \exp\left(b_{z'} + \mathbf{p}_u^\top \mathbf{q}_{z'}\right) \cdot \mathbf{q}_{z'}}{\sum_{z' \in c(z_{m-1})} \exp\left(b_{z'} + \mathbf{p}_u^\top \mathbf{q}_{z'}\right)} \right).
\end{aligned} \tag{21}$$

Here, $\mathbf{1}_{l+1}$ is an indicator function defined as

$$\mathbf{1}_{l+1} := \begin{cases} 1 & \text{if } n(i, l+1) = \text{left}(n(i,l)), \\ 0 & \text{otherwise.} \end{cases} \tag{22}$$

Before moving to the internal nodes, let us define another indicator function $\mathbf{1}_{z \in \text{path}_i}$ which is defined as

$$\mathbf{1}_{z \in \text{Path}_i} := \begin{cases} 1 & \text{if } z \text{ is in path}_i, \\ 0 & \text{other siblings nodes.} \end{cases} \tag{23}$$

Then, for each internal node $z \in \text{path}_i$ and its siblings, we have the gradient with respect to $\mathbf{q}_z$ as

$$\begin{aligned} \frac{\partial \mathcal{O}}{\partial \mathbf{q}_z} &= \sum_{l=1}^{L(z)} \frac{\partial \ln \Pr(z|u, \pi(z))}{\partial \mathbf{q}_z} + \sum_{\substack{m=1 \\ \forall z \in \Gamma}}^{M} \frac{\partial \ln \Pr\left(\mathbf{q}_{z_m} | \mathbf{q}_{z_{m-1}}, \sigma^2 \mathbf{I}\right)}{\partial \mathbf{q}_z} \\ &= \mathbf{1}_{z \in \text{path}_i} \cdot \mathbf{p}_u - \frac{\exp\left(b_z + \mathbf{p}_u^\top \mathbf{q}_z\right) \cdot \mathbf{p}_u}{\sum_{z' \in c(z_{m-1})} \exp\left(b_{z'} + \mathbf{p}_u^\top \mathbf{q}_{z'}\right)} \\ &\quad - \frac{\mathbf{q}_z - \mathbf{q}_{\pi(z)}}{\sigma^2} - \frac{\sum_{z' \in c(z)} (\mathbf{q}_z - \mathbf{q}_{z'})}{\sigma^2}. \end{aligned} \tag{24}$$

Moreover, we have the gradient with respect to bias b_z as

$$\frac{\partial \mathcal{O}}{\partial b_z} = \mathbf{1}_{z \in \text{Path}_i} - \frac{\exp\left(b_z + \mathbf{p}_u^\top \mathbf{q}_z\right)}{\sum_{z' \in c(z_{m-1})} \exp\left(b_{z'} + \mathbf{p}_u^\top \mathbf{q}_{z'}\right)}. \tag{25}$$

Finally, for each node level $l = \{1, 2, \ldots, L(i) - 1\}$ in the hierarchical softmax binary tree, we have the gradient with respect to $\mathbf{q}_{n(i,l)}$ and $b_{n(i,l)}$ as

$$\frac{\partial \mathcal{O}}{\partial \mathbf{q}_{n(i,l)}} = \left(\mathbf{1}_{l+1} - \sigma\left(y_{u,n(i,l)}\right)\right) \cdot \mathbf{p}_u \tag{26}$$

$$\frac{\partial \mathcal{O}}{\partial b_{n(i,l)}} = \mathbf{1}_{l+1} - \sigma\left(y_{u,n(i,l)}\right), \tag{27}$$

where $\mathbf{1}_{l+1}$ is the indicator function defined in Equation (22). With gradients with respect to $\Theta = \{\mathbf{p}_u, \mathbf{q}_z, \mathbf{q}_{n(i,l)}, b_z, b_{n(i,l)}\}$ being derived, we update Θ using stochastic gradient ascent rule $\Theta = \Theta + \epsilon \cdot \frac{\partial \mathcal{O}(\Theta)}{\partial \Theta}$. We summarize the parameter estimation procedure in Algorithm 1.

4.3. Complexity Analysis

Note that in each iteration our SUCM has a linear time complexity $O((\sum_{m=1}^{M} |L_m| + \log |c(z_M)|) \times |\mathcal{D}|)$, where $|\mathcal{D}|$ is the number of user–app adoption observations in the training dataset, $|L_m|$ is the number of categories or subcategories in the category hierarchy level m, and $\log |c(z_M)|$ is the logarithm of the number of apps under the most outside subcategory z_M, whose children nodes are apps. Therefore, the SUCM has the same complexity as the widely used latent factor models, which are usually linear to the number of observations $|\mathcal{D}|$. In most applications, value of $(\sum_{m=1}^{M} |L_m| + \log |c(z_M)|)$ will not be a large number. For example, in our app recommendation application with a dataset collected from Google Play, the worst case of $(\sum_{m=1}^{M} |L_m| + \log |c(z_M)|)$ is around 70.

ALGORITHM 1: Structural User Choice Model Estimation

```
Input: category tree $\Gamma$, user–app adoption observations $\mathcal{D} = \{(u, i)\}$, learning rate $\epsilon$.
Output: optimal $\Theta = \{\mathbf{p}_u, \mathbf{q}_z, \mathbf{q}_{n(i,l)}, b_z, b_{n(i,l)}\}$
begin
    for each most outside subcategory node $z_M$ do
        build a binary tree for hierarchical softmax
    end
    Initialize $\Theta$
    repeat
        sample a user–app adoption instance $\langle u, i, \text{path}_i \rangle$
        // update user latent factor
        $\mathbf{p}_u \leftarrow \mathbf{p}_u + \epsilon \cdot \frac{\partial \mathcal{O}}{\partial \mathbf{p}_u}$ (Equation (21))
        // update internal node latent factor
        for each internal node $z \in \text{path}_i$ and its siblings do
            $\mathbf{q}_z \leftarrow \mathbf{q}_z + \epsilon \cdot \frac{\partial \mathcal{O}(\Theta)}{\partial \mathbf{q}_z}$ (Equation (24))
            $b_z \leftarrow b_z + \epsilon \cdot \frac{\partial \mathcal{O}(\Theta)}{\partial b_z}$ (Equation (25))
        end
        // update hierarchical softmax binary tree node latent factor
        for for each node level $l = \{1, \ldots, L(i) - 1\}$ do
            $\mathbf{q}_{n(i,l)} \leftarrow \mathbf{q}_{n(i,l)} + \epsilon \cdot \frac{\partial \mathcal{O}(\Theta)}{\partial \mathbf{q}_{n(i,l)}}$ (Equation (26))
            $b_{n(i,l)} \leftarrow b_{n(i,l)} + \epsilon \cdot \frac{\partial \mathcal{O}(\Theta)}{\partial b_{n(i,l)}}$ (Equation (27))
        end
    until convergence or reach max_iter
    return $\hat{\Theta}$
end
```

5. EXPERIMENTS

This section presents an empirical evaluation of the performances of our model and previous methods. All the experiments are performed on a large-scale real-world app adoption dataset that we collected from Google Play.

5.1. Dataset Collection

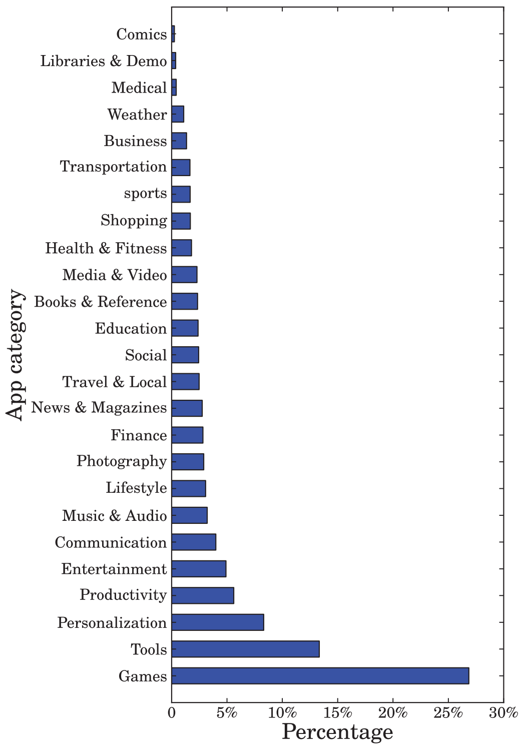
The Google Play is a centralized marketplace where all apps are organized in a predefined category tree. Apps are organized into 27 categories, and the category *Games* is further divided into 18 subcategories. Also Google Play has both free and paid apps. Users can review (i.e., rate or like) apps on Google Play. A user's review about apps he/she used are publicly available. Once we obtain the Google ID of a user, we can locate all apps the user has reviewed. Therefore, we first obtained a list of Google user IDs from the dataset shared from Gong et al. [2012] and wrote a crawler to collect the list of apps that had been reviewed by these users. For each retrieved app, we crawled its category and subcategory information from Google Play.

We treated a user having adopted an app if the review score, whose value is from one to five, is greater or equal to three. After excluding users who have adopted less than 40 apps to avoid cold start problem, we obtained a dataset with 52,483 users, 26,426 apps, and 3,286,156 review observations. The resulting user–app adoption matrix has a sparsity as high as 99.76% and each user adopts 62.61 apps on average, which is a very small fraction of all the apps. Table II shows some basic statistics of our dataset.

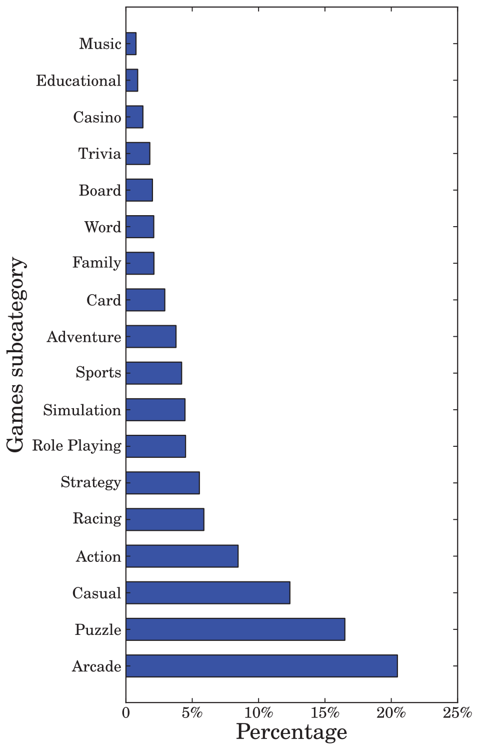
Since only 11.11% of all the apps in our dataset are paid apps, we do not distinguish between paid and free apps when constructing the hierarchical category tree Γ. The 26, 426 apps are categorized into 25 categories (the categories *Live Wallpaper*

Table II. Data Description

#users	#apps	#observations	sparsity
52,483	26,426	3,286,156	99.76%



(a) Category



(b) Games

Fig. 3. App distributions in the category hierarchy. (a) App distributions in app categories, and (b) app distributions in *Games* subcategory.

and *Widgets* defined by Google Play do not appear in our dataset). Figure 3 shows the detailed app distributions in different categories and the subcategories of Games. We observe that game apps take the highest percentage, accounting for 26.85% of all the apps; *Arcade* (20.46%), *Puzzle* (16.50%), and *Casual* (12.35%) are among the top three subcategories in *Games*, accounting for 49.31% of all the game apps.

5.2. Compared Approaches

We compare our SUCM with the following recommendation models.

—Logistic Latent Factor Model (LLFM) [Agarwal and Chen 2009]. LLFM was designed to model binary response using a cross-entropy loss function. In our problem, we adapt LLFM to solve the following optimization problem:

$$\arg\min_{\mathbf{P},\mathbf{Q},\mathbf{b}} \sum_{u,i\in\mathcal{D}} \ln\left(1+\exp\left(-\left(\mathbf{p}_u^\top \mathbf{q}_i + b_i\right)\right)\right) + \lambda_U \sum_{u\in\mathcal{U}} ||\mathbf{p}_u||_2^2 + \lambda_I \sum_{i\in\mathcal{I}} ||\mathbf{q}_i||_2^2 + \lambda_b \sum_{i\in\mathcal{I}} b_i^2,$$

where parameters λ_U, λ_I, and λ_b are regularization weights for users, items, and item bias, respectively.

—Probabilistic Matrix Factorization with negative samples (PMF$_{\mathrm{Neg}}$) [Salakhutdinov and Mnih 2008]. PMF is a standard latent factor model that is widely used for recommendations. We adapt PMF to our problem, i.e., we solve the following problem:

$$\arg\min_{\mathbf{P},\mathbf{Q},\mathbf{b}} \sum_{u,i\in\mathcal{D}} \left(y_{ui} - \left(\mathbf{p}_u^\top \mathbf{q}_i + b_i\right)\right)^2 + \lambda_U \sum_{u\in\mathcal{U}} ||\mathbf{p}_u||^2 + \lambda_I \sum_{i\in\mathcal{I}} ||\mathbf{q}_i||^2 + \lambda_b \sum_{i\in\mathcal{I}} b_i^2.$$

However, we only have positive adopted instance (u, i) that is treated as $y_{ui} = 1$. Thus, for each instance (u, i), we sample a certain number of negative instances $\{(u, j)\}$ and treat them as $y_{uj} = 0$. We denote this modified PMF as PMF$_{\mathrm{Neg}}$. Note that PMF$_{\mathrm{Neg}}$ is similar to the sample-based one-class collaborative filtering methods [Pan et al. 2008; Hu et al. 2008].

—*SVDFeature* [Chen et al. 2012]. SVDFeature is a feature-based latent factor model for recommendation settings with auxiliary information. We use category information as the auxiliary information for SVDFeature.

—*LibFM* [Rendle 2010, 2012]. LibFM is a software implementation for factorization machines (FM) [Rendle 2010] that models all interactions between variables (e.g., user, item, and auxiliary information). We also use category information as the auxiliary information and choose the two-way FM. One major difference between FM and SVDFeature is that SVDFeature only considers the interactions between user features and item features, whereas FM models all the interactions among all the available information.

—Bayesian Personalized Ranking (BPR) [Rendle et al. 2009]. BPR was first proposed to model personalized ranking with implicit feedback by treating observed user–item pairs as positive instances and sampling some of the unseen user–item pairs as negative instances. Given the preference triples $\mathcal{D} = \{(u, i, j)|i \succ j\}$, where $i \succ j$ indicates user u prefers item i than item i, BPR aims at maximizing the following optimization criterion:

$$\arg\max_{\mathbf{P},\mathbf{Q},\mathbf{b}} \left\{ \ln \prod_{(u,i,j)\in\mathcal{D}} \Pr((u,i,j)|i \succ_u j)\Pr(\Theta) \right\}$$
$$= \arg\max_{\mathbf{P},\mathbf{Q},\mathbf{b}} \left\{ \ln \prod_{(u,i,j)\in\mathcal{D}} \sigma(y_{ui} - y_{uj}|\Theta)\Pr(\Theta) \right\},$$

where $\sigma(\cdot)$ is the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, and $\Pr(\Theta)$ are Gaussian priors for the parameters.

—CCF [Yang et al. 2011]: Given an offer set, CCF models user–item choice behavior by encoding a local competition effect to improve recommendation performances. For each instance (u, i), we sample a certain number of negative instances to formulate the offer sets $\{(u, i, \mathcal{A}(i))$ as described in Yang et al. [2011]. Given collections of choice decision-making records $\mathcal{D} = \{(u, i, \mathcal{A}(i))\}$, CCF estimates the latent factors and the item bias terms by solving following optimization problem:

$$\arg\min_{\mathbf{P},\mathbf{Q},\mathbf{b}} \left\{ \sum_{(u,i,\mathcal{A}(i))\in\mathcal{D}} \ln\left[\sum_{j\in\mathcal{A}(i)} \exp\left(\mathbf{p}_u^\top \mathbf{q}_j + b_j\right)\right] - \left(\mathbf{p}_u^\top \mathbf{q}_i + b_i\right) \right.$$
$$\left. + \lambda_U \sum_{u\in\mathcal{U}} ||\mathbf{p}_u||_2^2 + \lambda_I \sum_{i\in\mathcal{I}} ||\mathbf{q}_i||_2^2 + \lambda_b \sum_{i\in\mathcal{I}} b_i^2 \right\}.$$

Implementations, training, and testing. All models are implemented with a stochastic gradient ascent/descent optimization method with an annealing procedure to discount learning rate ϵ at the iteration nIter with $\epsilon^{\text{nIter}} = \epsilon \frac{\nu}{\nu+\text{nIter}-1}$ by setting $\nu = 50$. The learning rate ϵ and the regularization weights are set by cross validation. All parameters are initialized by a Gaussian distribution $\mathcal{N}(0, 0.1)$. We randomly sample 80% of adopted apps of each user as the training dataset, and we use the remaining adopted apps for testing.

5.3. Evaluation Metrics

In this implicit feedback app recommendation setting, we present each user with N apps that have the highest predicted affinity values but are not adopted by the user in the training phase, and we evaluate different approaches based on which of these apps were actually adopted by the user in the test phase. More specifically, we adopt a variety of widely used metrics to evaluate different approaches. In the following, we elaborate each metric.

Precision and Recall Given a top-N recommendation list $C_{N,\text{rec}}$, precision and recall are defined as

$$\begin{aligned} \text{Precision@}N &= \frac{|C_{N,\text{rec}} \bigcap C_{\text{adopted}}|}{N} \\ \text{Recall@}N &= \frac{|C_{N,\text{rec}} \bigcap C_{\text{adopted}}|}{|C_{\text{adopted}}|}, \end{aligned} \tag{28}$$

where C_{adopted} are the apps that a user has adopted in the test data. The precision and recall for the entire recommender system are computed by averaging the precision and recall over all the users, respectively.

F-measure. F-measure balances between precision and recall. We consider the F_β metric, which is defined as

$$F_\beta = (1+\beta^2) \cdot \frac{\text{Precision} \times \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}, \tag{29}$$

where $\beta < 1$ indicates more emphasis on precision than recall. In our experiments, we use F_β metric with $\beta = 0.5$.

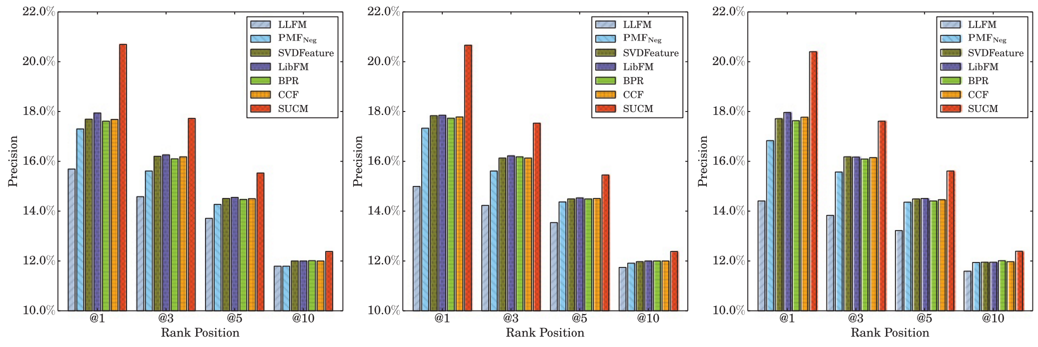
Mean Average Precision. Average precision (AP) is a ranked precision metric that gives larger credit to correctly recommended apps in higher positions. AP@N is defined as the average of precisions computed at all positions with an adopted app, namely,

$$\text{AP@}N = \frac{\sum_{k=1}^{N} P(k) \times \text{rel}(k)}{\min\{N, |C_{\text{adopted}}|\}}, \tag{30}$$

where $P(k)$ is the precision at cut-off k in the top-N list $C_{N,\text{rec}}$, and rel(k) is an indicator function equaling 1 if the app at rank k is adopted, otherwise zero. Finally, mean average precision (MAP@N) is defined as the mean of the AP scores for all users.

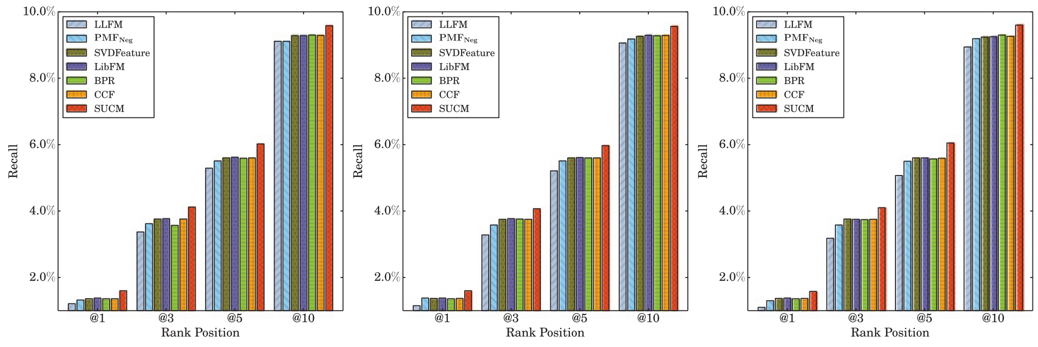
Normalized Discounted Cumulative Gain. NDCG is a ranked precision metric that gives larger credit to correctly recommended apps in higher positions. Specifically, the discounted cumulative gain (DCG) given a cut-off N is calculated by

$$\text{DCG}_{\text{N}} = \sum_{i=1}^{N} \frac{2^{\text{rel}_i} - 1}{\log_2(i+1)}, \tag{31}$$



(a) $K = 20$ (b) $K = 30$ (c) $K = 50$

Fig. 4. Precision @N with different latent dimensions K.



(a) $K = 20$ (b) $K = 30$ (c) $K = 50$

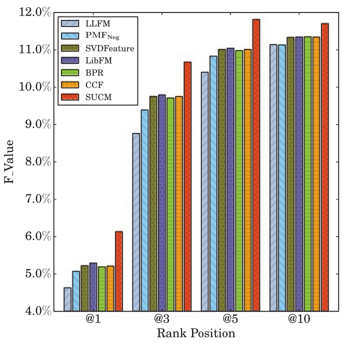
Fig. 5. Recall @N with different latent dimensions K.

where rel_i is the relevance score, which is binary. Then, the NDCG@N is computed as $\text{NDCG@N} = \frac{\text{DCG}_\text{N}}{\text{IDCG}_\text{N}}$, where IDCG_N is the DCG_N value of the ideal ranking list. The NDCG for the entire recommender system is computed by averaging the NDCG over all the users.

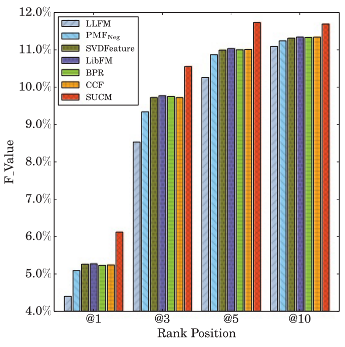
5.4. Performance Comparisons

In this subsection, we present the performance comparisons on top-N performances between our proposed SUCM and the baseline methods. We compare various approaches with three latent dimensions $K = 20$, $K = 30$, and $K = 50$, and four top-N values $N = 1, 3, 5, 10$.

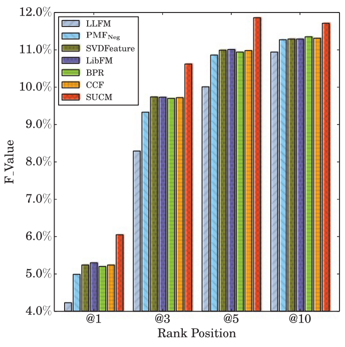
Figures 4, 5, and 6, respectively, show the precision@N, recall@N, and F_β@N of all compared approaches on our dataset. We find that our approach consistently and substantially outperforms the previous methods for different N and different K. Moreover, we observe that negative sampling-based methods PMF_Neg and BPR outperform LLFM that considers only positive instances for all the three considered number of latent dimensions. This is because LLFM polarizes toward the positive response values, and the learned recommendation model would predict positive for almost all unseen items and yield poor ranking performances. However, PMF_Neg and BPR mitigate the issue of LLFM via sampling unseen items as negative instances. Although PMF_Neg and BPR achieve close performances for top-1 recommendations, BPR works slightly better than



(a) $K = 20$



(b) $K = 30$



(c) $K = 50$

Fig. 6. F-measure F_β@N with different latent dimensions K ($\beta = 0.5$).

Table III. MAP@N with Different Latent Dimensions K

K	MAP	LLFM	PMF$_{Neg}$	SVDFeature	LibFM	BPR	CCF	SUCM
20	@1	15.69%	17.30%	17.69%	17.94%	17.61%	17.68%	**20.69%**
	@3	10.38%	11.29%	11.72%	11.82%	11.64%	11.69%	**13.40%**
	@5	8.17%	8.73%	9.00%	9.07%	8.95%	8.99%	**10.11%**
	@10	5.49%	5.69%	5.85%	5.88%	5.84%	5.85%	**6.40%**
30	@1	14.99%	17.33%	17.83%	17.85%	17.73%	17.78%	**20.66%**
	@3	10.07%	11.46%	11.71%	11.78%	11.72%	11.70%	**13.23%**
	@5	7.97%	8.89%	9.01%	9.05%	9.00%	9.00%	**9.97%**
	@10	5.38%	5.80%	5.85%	5.87%	5.86%	5.86%	**6.33%**
50	@1	14.41%	16.83%	17.71%	17.96%	17.63%	17.77%	**20.40%**
	@3	9.71%	11.23%	11.74%	11.77%	11.67%	11.72%	**13.25%**
	@5	7.71%	8.77%	9.01%	9.03%	8.94%	8.99%	**10.05%**
	@10	5.25%	5.76%	5.84%	5.86%	5.84%	5.85%	**6.36%**

PMF$_{Neg}$ for top-3, top-5, and top-10 recommendations. Moreover, CCF further slightly outperforms BPR in most cases. This is because CCF captures the local competition by a softmax of the chosen item over the offer set. Our proposed SUCM further improves upon CCF with significant margins for all the three evaluation metrics. For example, SUCM improves upon CCF with around 3% in terms of top-1 precision.

Besides, previous work [Chen et al. 2012; Rendle 2010, 2012] showed that latent factor-based recommendation could be improved by incorporating auxiliary information such as item features and context information. However, we observe that, by treating category information as auxiliary information, these methods (e.g., SVDFeature and LibFM) can only gain marginal improvements in terms of top N recommendation performances. Compared with counterpart method PMF$_{Neg}$ without auxiliary information, SVDFeature can only gain around 0.4% improvement and LibFM can only gain around 0.6% improvement in terms of top-1 precision, respectively. We argue that category information, treated as auxiliary feature, is not fine grained enough to discriminate user preferences. Quite differently, SUCM leverages the hierarchical structure of category information to better profile user interest preferences.

Precision, recall, and F-measure do not consider the ranking positions of correctly recommended apps. So we further adopt MAP and NDCG to provide more fine-grained understanding of these recommendation approaches. Intuitively, MAP and NDCG give larger credits to correctly recommended apps that are in higher ranking positions. Tables III and IV, respectively, show the MAP@N and NDCG@N of all compared

Table IV. NDCG@N with Different Latent Dimensions K

K	NDCG	LLFM	PMF_{Neg}	SVDFeature	LibFM	BPR	CCF	SUCM
20	@1	15.69%	17.30%	17.69%	17.94%	17.61%	17.68%	**20.69%**
	@3	14.83%	15.96%	16.54%	16.64%	16.45%	16.52%	**18.40%**
	@5	14.17%	14.94%	15.29%	15.36%	15.23%	15.28%	**16.72%**
	@10	12.69%	12.99%	13.27%	13.30%	13.26%	13.26%	**14.11%**
30	@1	14.99%	17.33%	17.83%	17.85%	17.73%	17.78%	**20.66%**
	@3	14.41%	16.00%	16.52%	16.59%	16.54%	16.52%	**18.24%**
	@5	13.89%	15.04%	15.29%	15.34%	15.28%	15.30%	**16.61%**
	@10	12.53%	13.10%	13.26%	13.29%	13.27%	13.28%	**14.07%**
50	@1	14.41%	16.83%	17.71%	17.96%	17.63%	17.77%	**20.40%**
	@3	13.96%	15.87%	16.54%	16.58%	16.47%	16.52%	**18.23%**
	@5	13.51%	14.96%	15.28%	15.33%	15.21%	15.26%	**16.68%**
	@10	12.29%	13.08%	13.24%	13.27%	13.26%	13.25%	**14.07%**

approaches. Again, we observe consistent and substantial improvements of our SUCM upon previous methods.

Summary. Through extensive evaluations, we found that our method SUCM consistently and substantially outperforms previous methods in terms of a variety of evaluation metrics. We argue that SUCM achieves this performance gain by learning fine-grained user preferences via leveraging the hierarchical category tree of apps and capturing the competitions between apps.

6. RELATED WORK

Our work is related to two research fields, personalized recommendation methodology and mobile app recommendation.

Recommendation methodology. The most popular model-based approaches are based on the latent factor models [Salakhutdinov and Mnih 2008; Koren et al. 2009; Agarwal and Chen 2009; Wu et al. 2016]. For the binary implicit feedback setting, models such as LLFM use cross-entropy loss [Agarwal and Chen 2009], but it is still apt to obtain an estimator that would polarize toward the positive response values, thus leading to limited top N performances. Negative sampling provides an alternative by sampling a certain number of unseen items as negative samples. Then, standard latent factor models such as PMF [Salakhutdinov and Mnih 2008] can be adopted. Hu et al. [2008] proposed to treat implicit data as indication of positive and negative preferences associated with vastly varying confidence levels on the objective function. Pan et al. [2008] used a similar strategy by applying weighted low rank approximation. Instead of optimizing point-wise loss function, BPR [Rendle et al. 2009] optimizes a pairwise loss function to preserve the relative order of items for each user.

There are few works that adopt discrete choice models to model user–item choices for recommendation [Yang et al. 2011] and for geographic ranking [Kumar et al. 2015]. Discrete choice models [Luce 1959; McFadden 1973] are built on established theories on consumer preferences and utility and have been widely used for understanding consumer behavior in different application domains, such as travel [Ben-Akiva and Lerman 1985], transportation [Train 1978], and brand choice [Guadagni and Little 1983]. Based on a discrete choice model, Yang et al. [2011] proposed a CCF model to learn user–item choice to improve recommendation performance. Given users' interaction with offer sets, CCF models user choice by a softmax function of the chosen item over the offer set. In this sense, CCF can also be categorized into the sampling-based method with samples given in the offer set. Though trying to model user choice process, CCF does not consider the structural dependence between items to be chosen for users; thus, the choice model in CCF is "flat" rather than structural. We extend the "flat" choice model into SUCM to capture fine-grained user preferences for mobile app recommendation.

Recently, there are a few works to explore the item hierarchy and side information for recommendation. For instance, Kanagal et al. [2012] and Ahmed et al. [2013] proposed to learn user preferences with additional hierarchical item relationship and other side information such as brand and temporal purchase sequence. Instead of using the predefined item hierarchy, some other works also try to learn the item taxonomy [Zhang et al. 2014a]. However, these works did not consider the competitions among similar items or among similar categories/subcategories. We do not compare our methods with them because they utilized more side information such as item brands and item semantics, but it is an interesting future work for us to extend our framework to incorporate similar side information in the domain of app recommendation. Besides, Ziegler et al. [2005] utilized taxonomy information to balance and diversify personalized recommendation lists. However, our work different from Ziegler et al. [2005] in both the purpose and the way of utilizing taxonomy information.

Mobile app recommendation. Recently, app recommendation has drawn an increasing number of attentions. Different from other domains such as movies [Bell and Koren 2007], musics [Aizenberg et al. 2012], and point-of-interests [Liu et al. 2013, 2015b], app recommendation has its own characteristics. Yin et al. [2013] considered a tradeoff between satisfaction and temptation for app recommendation with a special focus on the case that a user would like to replace an old app with a new one. Similarly, [Lin et al. 2014] and [Lin 2014] considered app versions to improve app recommendation by incorporating features distilled from version descriptions. Karatzoglou et al. [2012] provided a context-aware recommendation using tensor factorization by including context information such as location, moving status, and time. Woerndl et al. [2007] applied a hybrid method for context-aware app recommendation. To address the cold-start problem for app recommendation, [Lin et al. 2013] and [Lin 2014] proposed to leverage side information from Twitter. Specifically, information of followers of an app's official Twitter account is collected and utilized to model the app, providing an estimation about which users may like the app. Davidsson et al. [2011] presented a context-based recommender prototype for cold-start user users. Zhu et al. [2014] proposed a mobile app ranking system by considering both the app's popularity and security risks. More recently, Liu et al. [2015a] studied personalized app recommendation by reconciling user functionality preferences and user privacy preferences. Baeza-Yates et al. [2015] proposed a method to predict which app a user is going to use by leveraging spatio-temporal context features, and Park et al. [2015] proposed a method to improve the accuracy of mobile app retrieval by jointly modeling app descriptions and user reviews using topic model. However, these works are orthogonal to ours because they use other auxiliary information such as app versions, app satisfaction and temptation, and app privacy, whereas our work focuses on leveraging app taxonomy to model structural user choices among competing apps.

7. CONCLUSION AND FUTURE WORK

In this article, we proposed a novel SUCM to learn fine-grained user preferences via leveraging the tree hierarchy of apps and capturing competitions between apps for app recommendation. Specifically, given all apps in a centralized mobile app market organized as a category tree, we represented the structural user choice as a unique choice path, starting from the root till the leaf node where user makes an app adoption decision, over the category hierarchy. Then, we captured the structural choice procedure by cascading user preferences over the choice path through a novel probabilistic model. We also designed an efficient learning algorithm to estimate the model parameters. Moreover, we collected a real-world large-scale user–app adoption dataset from Google Play and used it to evaluate our method and various previous methods. Our results demonstrated that our method achieved consistent and substantial performance improvements over previous methods.

There are a few interesting future directions that are worth exploring. (1) Human-induced taxonomies are usually noisy and incomplete, and they do no evolve with a change in user demographics or product inventory. Zhang et al. [2014a] proposed a probabilistic model that is able to automatically discover the taxonomies from online shopping data. An interesting future work is to explore a unified model that could jointly learn the taxonomy and the structural choice model. (2) There are plenty of user reviews in the Google Play store. Several previous works [Zhang et al. 2014b; Wu and Ester 2015; Chen et al. 2016] have shown that user reviews provide more detailed information on why a user gives an item a specific rating. Integrating our model with user reviews helps us better understand fine-grained user preferences. (3) We are also interested in adapting our model to other domains in which items are also hierarchically categorized.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the article.

REFERENCES

Deepak Agarwal and Bee-Chung Chen. 2009. Regression-based latent factor models. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09)*. 19–28.

Amr Ahmed, Bhargav Kanagal, Sandeep Pandey, Vanja Josifovski, Lluis Garcia Pueyo, and Jeff Yuan. 2013. Latent factor models with additive and hierarchically-smoothed user preferences. In *Proceedings of the 6th ACM International Conference on Web Search and Data Mining (WSDM'13)*. 385–394.

Natalie Aizenberg, Yehuda Koren, and Oren Somekh. 2012. Build your own music recommender by modeling internet radio streams. In *Proceedings of the 21st International Conference on World Wide Web*. 1–10.

Ricardo Baeza-Yates, Di Jiang, Fabrizio Silvestri, and Beverly Harrison. 2015. Predicting the next app that you are going to use. In *Proceedings of the 8th ACM International Conference on Web Search and Data Mining (WSDM'15)*. 285–294.

Robert M. Bell and Yehuda Koren. 2007. Lessons from the Netflix prize challenge. *SIGKDD Explor. Newsl.* 9, 2 (Dec. 2007), 75–79.

Moshe E. Ben-Akiva and Steven R. Lerman. 1985. *Discrete Choice Analysis: Theory and Application to Travel Demand*. Vol. 9. MIT Press.

Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT 2010*. 177–186.

Tianqi Chen, Weinan Zhang, Qiuxia Lu, Kailong Chen, Zhao Zheng, and Yong Yu. 2012. SVDFeature: A toolkit for feature-based collaborative filtering. *J. Mach. Learn. Res.* 13, 1 (Dec. 2012), 3619–3622.

Xu Chen, Yongfeng Zhang, Tao Xu, and Zheng Qin. 2016. Learning to rank features for recommendation over multiple categories. In *Proceedings of the 39th International Acm SIGIR Conference on Research & Development in Information Retrieval*. ACM.

Christoffer Davidsson and Simon Moritz. 2011. Utilizing implicit feedback and context to recommend mobile applications from first use. In *Proceedings of the 2011 Workshop on Context-awareness in Retrieval and Recommendation (CaRR'11)*. 19–22.

Neil Zhenqiang Gong, Wenchang Xu, Ling Huang, Prateek Mittal, Emil Stefanov, Vyas Sekar, and Dawn Song. 2012. Evolution of social-attribute networks: Measurements, modeling, and implications using google+. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*. ACM, 131–144.

Peter M. Guadagni and John D. C. Little. 1983. A logit model of brand choice calibrated on scanner data. *Mark. Sci.* 2, 3 (1983), 203–238.

Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *Proceedings of 8th IEEE International Conference on Data Mining, (ICDM'08)*. 263–272.

Bhargav Kanagal, Amr Ahmed, Sandeep Pandey, Vanja Josifovski, Jeff Yuan, and Lluis Garcia-Pueyo. 2012. Supercharging recommender systems using taxonomies for learning user purchase behavior. *Proc. VLDB Endow.* 5, 10 (June 2012), 956–967.

Alexandros Karatzoglou, Linas Baltrunas, Karen Church, and Matthias Böhmer. 2012. Climbing the app wall: Enabling mobile app discovery through context-aware recommendations. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM'12)*. 2527–2530.

Yehuda Koren. 2008. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08)*. 426–434.

Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (Aug. 2009), 30–37.

Ravi Kumar, Mohammad Mahdian, Bo Pang, Andrew Tomkins, and Sergei Vassilvitskii. 2015. Driven by food: Modeling geographic choice. In *Proceedings of the 8th ACM International Conference on Web Search and Data Mining (WSDM'15)*. 213–222.

Jovian Lin. 2014. *Mobile App Recommendation*. Ph.D. Dissertation. National University of Singapore.

Jovian Lin, Kazunari Sugiyama, Min-Yen Kan, and Tat-Seng Chua. 2013. Addressing cold-start in app recommendation: Latent user models constructed from Twitter followers. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'13)*. 283–292.

Jovian Lin, Kazunari Sugiyama, Min-Yen Kan, and Tat-Seng Chua. 2014. New and improved: Modeling versions to improve app recommendation. In *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'14)*. 647–656.

Bin Liu, Yanjie Fu, Zijun Yao, and Hui Xiong. 2013. Learning geographical preferences for point-of-interest recommendation. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*. 1043–1051.

Bin Liu, Deguang Kong, Lei Cen, Neil Zhenqiang Gong, Hongxia Jin, and Hui Xiong. 2015a. Personalized mobile app recommendation: Reconciling app functionality and user privacy preference. In *Proceedings of the 8th International Conference on Web Search and Data Mining (WSDM'15)*. 315–324.

Bin Liu, Hui Xiong, Spiros Papadimitriou, Yanjie Fu, and Zijun Yao. 2015b. A general geographical probabilistic factor model for point of interest recommendation. *IEEE Trans. Knowl. Data Eng.* 27, 5 (2015), 1167–1179.

R. Duncan Luce. 1959. *Individual Choice Behavior: A Theoretical Analysis*. Wiley.

Charles F. Manski. 1977. The structure of random utility models. *Theory Decis.* 8, 3 (1977), 229–254.

Daniel McFadden. 1973. Conditional logit analysis of qualitative choice behavior. In *Frontiers in Econometrics*, P. Zarembka (Ed.). Academic Press, New-York.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv:1301.3781* (2013).

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Proceedings of Advances in Neural Information Processing Systems*. 3111–3119.

Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*. 246–252.

Rong Pan, Yunhong Zhou, Bin Cao, Nathan Nan Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. 2008. One-class collaborative filtering. In *Proceedings of 8th IEEE International Conference on Data Mining (ICDM'08)*. 502–511.

Dae Hoon Park, Mengwen Liu, ChengXiang Zhai, and Haohong Wang. 2015. Leveraging user reviews to improve accuracy for mobile app retrieval. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'15)*. 533–542.

Steffen Rendle. 2010. Factorization machines. In *Proceedings of IEEE 10th International Conference on Data Mining (ICDM)*. IEEE, 995–1000.

Steffen Rendle. 2012. Factorization machines with libFM. *ACM Trans. Intell. Syst. Technol.* 3, 3, Article 57 (May 2012), 22 pages. DOI:http://dx.doi.org/10.1145/2168752.2168771

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI'09)*. 452–461.

Ruslan Salakhutdinov and Andriy Mnih. 2008. Probabilistic matrix factorization. In *Proceedings of Neural Information Processing Systems (NIPS)*, Vol. 20.

Kenneth Train. 1978. A validation test of a disaggregate mode choice model. *Transp. Res.* 12, 3 (1978), 167–174.

Wolfgang Woerndl, Christian Schueller, and Rolf Wojtech. 2007. A hybrid recommender system for context-aware recommendations of mobile applications. In *Proceedings of IEEE 23rd International Conference on Data Engineering Workshop, 2007.* IEEE, 871–878.

Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the 9th ACM International Conference on Web Search and Data Mining (WSDM'16)*. 153–162.

Yao Wu and Martin Ester. 2015. FLAME: A probabilistic model combining aspect based opinion mining and collaborative filtering. In *Proceedings of the 8th ACM International Conference on Web Search and Data Mining (WSDM'15)*. ACM, 199–208.

Shuang-Hong Yang, Bo Long, Alexander J. Smola, Hongyuan Zha, and Zhaohui Zheng. 2011. Collaborative competitive filtering: Learning recommender using context of user choice. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'11)*. 295–304.

Peifeng Yin, Ping Luo, Wang-Chien Lee, and Min Wang. 2013. App recommendation: A contest between satisfaction and temptation. In *Proceedings of the 6th ACM International Conference on Web Search and Data Mining (WSDM'13)*. 395–404.

Yuchen Zhang, Amr Ahmed, Vanja Josifovski, and Alexander Smola. 2014a. Taxonomy discovery for personalized recommendation. In *Proceedings of the 7th International Conference on Web Search and Data Mining (WSDM'14)*. 243–252.

Yongfeng Zhang, Guokun Lai, Min Zhang, Yi Zhang, Yiqun Liu, and Shaoping Ma. 2014b. Explicit factor models for explainable recommendation based on phrase-level sentiment analysis. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, 83–92.

Hengshu Zhu, Hui Xiong, Yong Ge, and Enhong Chen. 2014. Mobile app recommendation with security and privacy awareness. In *Proceedings of the 20th ACM International Conference on Knowledge Discovery and Data Mining (KDD'14)*.

Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. 2005. Improving recommendation lists through topic diversification. In *Proceedings of the 14th International Conference on World Wide Web (WWW'05)*. 22–32.

Received August 2015; revised May 2016; accepted August 2016