Identifying Functional Aspects From User Reviews for Functionality-Based Mobile App Recommendation

Xiaoying Xu
Department of Decision Science, School of Business Administration, South China University of Technology, 381 Wushan Road, Tianhe District, Guangzhou, 510000, P.R. China. E-mail: bmxyxu@scut.edu.cn

Kaushik Dutta
Department of Information Systems and Decision Sciences, Muma College of Business, University of South Florida, 4202 E. Fowler Avenue, BSN 3403, Tampa, FL 33620, USA. E-mail: duttak@usf.edu

Anindya Datta
Mobilewalla Inc, 03-20 Franklin 3, Science Park Drive, Singapore, 118223. E-mail: adatta@mobilewalla.com

Chunmian Ge
Department of Financial Management, School of Business Administration, South China University of Technology, 381 Wushan Road, Tianhe District, Guangzhou, 510000, P.R. China. E-mail: bmgecm@scut.edu.cn

The explosive growth of mobile apps makes it difficult for users to find their needed apps in a crowded market. An effective mechanism that provides high quality app recommendations becomes necessary. However, existing recommendation techniques tend to recommend similar items but fail to consider users' functional requirements, making them not effective in the app domain. In this article, we propose a recommendation architecture that can generate app recommendations at the functionality level. We address the redundant recommendation problem in the app domain by highlighting users' functional requirements, an element that has received scant attention from existing recommendation research. Another main feature of our work is extracting app functionalities from textural user reviews for recommendation. We also propose an effective approach for functionality extraction. Experiments conducted on a real-world dataset show that our proposed *AppRank* method outperforms other commonly used recommendation methods. In particular, it doubles the recall value of the second best method under an extremely sparse setting, increases the overall ranking accuracy of the second best method by 14.27%, and retains a high diversity of 0.99.

Received July 1, 2016; revised May 27, 2017; accepted June 20, 2017

 DOI: 10.1002/asi.23932

Introduction

Accelerated by the popularity of smart devices, the mobile application (or app for short) market is growing explosively. For instance, the Apple App Store provides more than one million apps in 24 categories for users in 155 countries around the world.[1] Although such growth has created huge economic opportunities for app developers and technology companies, and has also brought great convenience to users by providing countless useful apps, the market is facing a big problem of "information overload." On one hand, tens of thousands of new apps are continuously being released in app stores, but most of them remain unknown to users searching via keyword. on the other hand, users struggle to find the apps they need in such crowded app stores (B. Liu et al., 2015; Q. Liu, Ma, Chen, & Xiong, 2013). Therefore, effective mechanisms that enable precise app marketing and facilitate new app discovery, are essential for both developers and users.

To alleviate the information overload problem, many industry solutions have been proposed in traditional E-Commence domains. Among them, Recommender Systems (RSs) stand out as the most widely used technique in practice. According to Hosanagar, Fleder, Lee, and Buja (2013), 35% of Amazon's sales and 60% of Netflix's sales originate from their RSs. It is clear how important RSs are to the

[1]http://www.apple.com/pr/library/2014/01/07App-Store-Sales-Top-10-Billion-in-2013.html

retailers. Even a small improvement in the accuracy of RSs would make a big difference to their revenues. Although the general goal of mobile app recommendation is similar to goals in traditional domains—to guide users to items that are relevant to their interests, mobile apps have unique features that make the solutions in traditional domains less effective in the app domain.

One of the most important characteristics of mobile app selection is that it is based more on a combination of the users' functional and nonfunctional requirements for mobile apps, than on the users' general tastes. The functional aspect of user requirements refers to the functionalities provided by the apps and needed by the user, such as *weather forecast* and *navigation*; the nonfunctional aspect is more about the quality-relevant features of the apps, such as ease of use, UI design, power consumption, etc. To generate accurate app recommendations, it is very important to differentiate and fully consider the two types of user requirements, but this has not attracted sufficient attention from recommendation research.

The most widely used recommendation techniques in traditional domains: Collaborative Filtering (CF; Sarwar, Karypis, Konstan, & Reidl, 2001) and Content-based Filtering (CB; Pazzani & Billsus, 2007), usually generate recommendations based on user ratings. In the app domain, however, rating values indicate more about users' evaluation of the nonfunctional aspects of the app, but can hardly reflect the users' functional requirements. For example, even if a user gives a very low rating to an app providing the weather forecast, we can only say the user is not satisfied with this app (maybe because it consumes too much power), but we cannot deny the fact that this user needs the functionality of weather forecast. When applied in the app domain, traditional techniques fail to reveal the detailed functionalities inside apps, and lack the ability to capture users' functional requirements, which may worsen the accuracy of recommendations.

Moreover, most existing techniques tend to recommend similar items, so the diversity of the recommendations generated by these methods is usually low (Adomavicius & Tuzhilin, 2005). A similar-recommendation strategy may work in certain domains. For instance, a user who likes the movie *Titanic* may be glad to watch another romantic movie similar to *Titanic*; however, if a user has installed an app providing particular functionality, for example, weather forecast, he or she needs no more similar apps with the same functionality of weather forecast, unless they provide additional functionalities. If existing recommendation techniques are directly applied in the app domain without paying attention to users' functional requirements, users may end up with receiving a mass of redundant app recommendations providing similar functionalities.

Recent research has paid increased attention to mobile app recommendation. However, the functional aspects of user requirements for mobile apps have received scant attention from researchers, and there has been no reported work on mobile app recommendation trying to address the redundancy problem. To bridge this gap, in this research, we extend our previous work (Xu, Dutta, & Datta, 2014), and propose a functionality-based recommendation solution that is able to provide more accurate and more diverse app recommendations by drilling down into the detailed functionalities of the apps.

Our work is among the first to consider the functional aspect of user requirements in mobile app recommendation. To achieve this goal, we propose extracting app functionalities from textual user reviews and develop a graph-based approach that enables functionality-based app recommendation. The results of experiments conducted on a real-world mobile app data set show that our proposed method outperforms other commonly used methods in terms of stability against data sparsity, ranking accuracy in top N recommendations, overall ranking correctness and recommendation diversity. In particular, our method doubles the recall value of the second-best method under an extremely sparse setting, increases the overall ranking accuracy of the second-best method by 14.27%, and retains a high diversity of 0.99.

The remainder of this article is organized as follows: first we review related research. Next, we describe the intuition behind our proposed solution and provide an overview followed by a more detailed elaboration. Then we evaluate our solution and present the results of our evaluation. Finally, we discuss the contribution of our work to the field and possibilities for further work.

Related Work

Recently, researchers have started paying attention to mobile app recommendation, and an increasing amount of research on app recommendation is being done. In the following, we will review related work on both traditional consumer product domains and mobile app domain. We also discuss related work on page-rank based methods, which will be adapted in our method to discover new functionalities for users.

Traditional Recommendation

The recommendation research in traditional consumer product domains mostly belongs to two main streams, that is, CF and CB.

CF has been explored in-depth during the last 10 years and represents the most popular recommendation algorithm. The basic assumption of CF is that people who have shared the same taste in some items in the past are more likely to feel similarly towards other items in the future. Following this assumption, two types of CF algorithms have been proposed, that is, user-based (UCF; Resnick, Iacovou, Suchak, Bergstorm, & Riedl, 1994) and item-based (ICF; Sarwar et al., 2001), both of which work on the same basic principal. First the neighborhood relationship is built by measuring the similarity (typically cosine similarity or Pearson correlation) between users or items based on their rating vectors, then the average ratings from the neighborhoods weighted by their similarities are used to predict the missing ratings. To alleviate the data sparsity problem that is a major

DOI: 10.1002/asi

limitation of the traditional CF methods, latent factor models such as Nonnegative Matrix Factorization (NMF; Lee & Seung, 1999) and Singular Value Decomposition (SVD; Paterek, 2007) comprise alternative approaches to CF by transforming both items and users to the same latent factor space, which try to explain ratings by characterizing both users and items on the latent factors that are automatically inferred from user feedback. CF methods are rating-centric and lack the ability to capture users' functional requirements when applied in apps domain, which is a significant drawback addressed by our method.

CB recommends items that are similar to what the target user has liked in the past. In a typical CB system, the interests of every user are represented as a structured profile that is constructed by analyzing the content of items rated by this user, and recommendations are generated by comparing the content similarities between item attributes and user profiles (Pazzani & Billsus, 2007). CB generates recommendations based on item content; therefore, it has an advantage in recommending new items for which no user preference data is available. CB works well in traditional domains like books, music, movies etc. However, in the apps domain, CB is no longer effective because users do not need similar apps with overlapping functionalities. Although our proposed method also employs content information of the apps (i.e., apps' functionalities), the logic of generating recommendations is different. We emphasize users' functional requirements and aim at recommending apps with new functionalities needed by the users, addressing the redundancy problem.

We also note that there are a few studies that develop utility-based recommender systems, for example, Spiekermann and Paraschiv (2002), Yi and Deng (2009), and Felfernig, Mandl, Schippel, Schubert, and Teppan (2010). Such systems recommend items by constructing a utility-function to match users' stated preferences over all attributes of the items. The benefit of these approaches is that many nonproduct features related to utility, such as delivery time, can be incorporated in the recommendation process. However, such systems are limited by the need for high user involvement (Burke, 2002; Pu, Faltings, Chen, Zhang, & Viappiani, 2011). Users are required to input the utility function by looking at, selecting and weighing each feature of the preferred items, or to fill out interactive questionnaires, before going into recommendations. Although our proposed method also tries to maximize the utility of the recommended apps, the underlying mechanism is totally different. Instead of asking the users to explicitly input the utility function, our method automatically infers users' requirements by constructing and analyzing the Functionality Graph, without user involvement. Therefore, our approach is more feasible than the utility-based systems for dealing with a huge number of app functionalities.

Mobile App Recommendation

A few studies propose extending traditional recommendation algorithms and adapting them to the app domain. For example, *AppJoy* (Yan & Chen, 2011) replaces the user ratings in traditional RSs with usage scores composed by recency, frequency and duration, and then performs item-based CF recommendation. Lin, Sugiyama, Kan, and Chua (2013) propose extending the model-based RS by constructing latent user models from apps' twitter followers, addressing the cold-start problem of app recommendation. B. Liu et al. (2015) consider users' privacy preferences in personalized app recommendations. Hybrid methods also exist. For example, Xu et al. (2014) report a multiobject approach to evolve existing mobile app RSs. Although these solutions have proven to be effective to some extent in recommending apps, they do not consider much about the unique characteristics of apps.

Noticing this limitation, some researchers have shifted their focus to a unique characteristic of mobile apps—context, and a few context-aware systems have been proposed in the app domain. Such systems record users' context information, for example, physical location, at a particular time and then enhance app recommendation by exploiting the collected context information (Q. Liu et al., 2013). For example, Böhmer, Bauer, and Krüger (2010) explored the design space for context-aware app recommendation and developed a prototype app RS on the Android platform called *Appazaar*. The *Djinn* model introduced by Karatzoglou, Baltrunas, Church, and Böhmer (2012) utilizes the user-app-context relationship using tensor factorization, providing a new context-aware CF approach for app recommendation. Shi et al. (2012) also apply tensor factorization to integrate implicit feedback data with contextual information, and they propose generating app recommendations by optimizing the ranking (i.e., MAP). Context-aware app RSs are highlighted because they consider one important feature of mobile app, that is, context information. Such systems show better performance than traditional methods in recommending apps. However, context information is very difficult to collect, because of privacy concerns and other constraints. It has been a significant limitation of context-aware systems.

To conclude, existing works on mobile app recommendation do consider some unique features in the app domain; however, no reported work has been found which both recommends apps at the functionality level and avoids redundant recommendations. These gaps will be addressed with our proposed method.

PageRank-Based Methods

PageRank (Page, Brin, Motwani, & Winograd, 1999), a graph-based ranking algorithm proposed by Google, has been successfully applied in analyzing the link-structure of the World Wide Web. The objective of PageRank is to determine the importance of a given webpage on the web hyperlink structure. The basic assumption of PageRank is that a web page is more likely to be authoritative if it is linked to by many other authoritative pages. The implementation of PageRank is based on a "voting" mechanism. If a webpage links to another page, it denotes a vote to that

DOI: 10.1002/asi

target page. Moreover, the weight of the vote is determined by the importance of the webpage which gives the vote. Finally, the greater the weight of the vote a webpage receives, the more important it is. The final weight, that is, the PageRank score, of a webpage is determined by a random walk process which iterates the voting process throughout each node in the graph until it converges.

Based on PageRank, many variants in different domains have been proposed. For example, Mihalcea and Tarau (2004) propose a graph-based ranking model called TextRank for keyword and sentence extraction in the domain of natural language processing. In the TextRank model, each word is modelled as a vertex, and the edges in graph represent the concurrence of words in the document. Jeh and Widom (2003) introduce the personalized PageRank vector into the original model and propose a personalized version of PageRank, which is able to capture user preference. FolkRank, proposed by Hotho, Jäschke, Schmitz, Stumme, and Althoff (2006), is an adaption of the PageRank algorithm for folksonomy ranking and searching. FolkRank employs a differential approach to compute a FolkRank score by taking the difference between the personalized PageRank score and the original PageRank score.

Graph-based approaches, such as PageRank, are good at exploring transitive associations in the data (Huang, 2004). The potential value of transitive associations for the data sparsity problem in recommender systems has been highlighted by many researchers in this field (Guan, Bu, Mei, Chen, & Wang, 2009; Huang, 2004; Kim & El Saddik, 2011). In our context, generally, users install only a small number of apps on their devices, making the usage data relatively sparse. With the help of graph-based models, the sparsity problem can be alleviated by utilizing the transitive association among the functional aspects of the apps. For this purpose, our proposed method combines and adapts TextRank and FolkRank in the context of app functionality prediction, and we call it *AppRank*. The details of our adaption will be provided in the ensuing sections.

Intuition and Overview

We are interested in helping mobile app users discover new functionalities they may need, and recommending apps that can truly meet their requirements. Our proposed method is motivated by users' real-life behavior of selecting mobile apps. When choosing an app to install, a user usually first considers whether the app provides the functionalities he or she needs by reading the app's description. If there are many alternatives providing similar functionalities, the user may try each of them and evaluate them on other nonfunctional aspects (e.g., UI design, ease of use, power consumption), and then select the most preferred one to use.

At a high level, our method automates this process through three main steps: (a) knowing all the functionalities provided by the apps that a user has been using; (b) predicting what other functionalities this user may need; and (c) helping the user select better apps providing these desired functionalities. One of the most outstanding features that differentiate our method from existing works is that we generate recommendations at the functionality level, truly capturing users' functional requirements.

To achieve our goal, the most important problem we need to solve is obtaining the functionalities of each app. An intuitive solution is to extract app functionalities from their textual descriptions. App descriptions have the advantage of using more formal vocabulary and containing less noisy content, but we quickly realize that descriptions are short texts wherein functionalities may not be repeatedly stated. Most of the traditional keyword extraction techniques (usually based on term frequency) are designed for long articles, which may not be effective when applied to app descriptions.

Fortunately, researchers have found that item features are frequently mentioned in customer reviews. This motivates us to obtain app functionalities from user reviews. However, user reviews often contain a lot of noisy content that is not relevant to the app functionalities. To address this problem, we make good use of the app descriptions, from which we propose to construct a functionality vocabulary to filter out the noisy content in the user reviews. Then we perform frequency analysis on the user reviews, which helps to extract high quality feature words and phrases related to the app functionalities.

Next, after acquiring the app functionalities, we propose a graph-based ranking method to discover new functionalities for the users. A functionality cooccurrence graph which captures the global functionality usage patterns of all users is constructed, and functionality prediction is performed as spreading activation (Huang, 2004). A two-stage PageRank approach can be employed to implement this activation-spreading process: for each user, a small subset of nodes representing the functionalities in use are activated with high weights, and other potential associated functionality nodes are then activated by receiving the weights propagated along the functionality cooccurrence graph in the PageRank process.

We also intelligently filter out apps with overlapping functionalities by a competition mechanism, therefore capturing user requirements and addressing the redundancy problem. The details of our proposed solution will be introduced in the following section.

Solution Details

In this section, we will first show the architecture of our proposed solution, followed by the details of each component in the architecture.

Our proposed app recommendation architecture is shown in Figure 1. There are the following three main components in the architecture: *App Data Crawler*, *Functionality Extractor*, and *App Recommender*. We use the App Data Crawler to collect app descriptions and corresponding user reviews. From the collected data, app functionalities are then extracted by the Functionality Extractor. Finally, the App Recommender predicts new functionalities for the user, selects candidate apps to recommend, and intelligently filters

DOI: 10.1002/asi

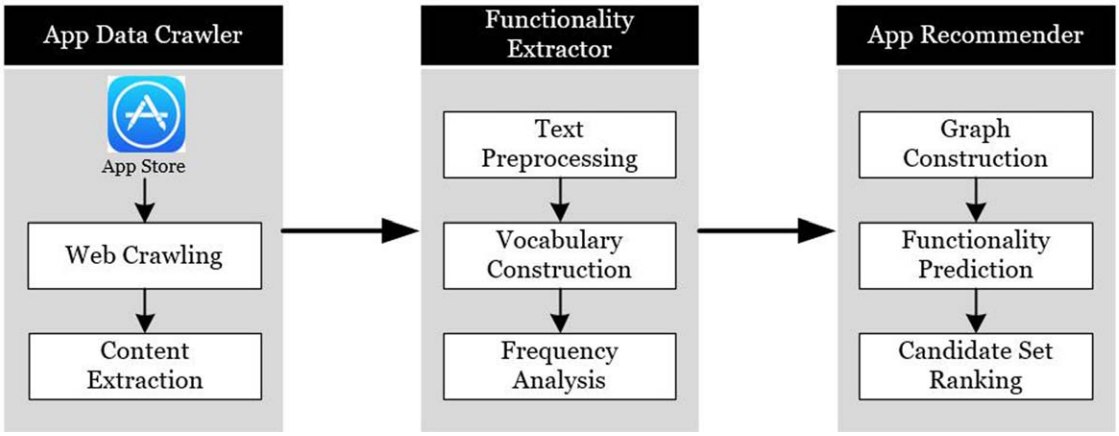

FIG. 1. App recommendation architecture. [Color figure can be viewed at wileyonlinelibrary.com]

out apps with overlapping functionalities. More details of each component will be given in the ensuing sections.

App Data Crawler

The main task of the crawler is to collect web pages containing app descriptions and user reviews from the app store. Figure 2 shows one of the app web pages. Because the needed content is embedded in HTML files, we develop an extractor to extract the textual content of app descriptions and user reviews. User ratings associated with reviews are also isolated.

Functionality Extractor

Text preprocessing. The inputs of the Functionality Extractor are the textual content of each app's descriptions and user reviews. We use the Stanford Core Natural Language Processing toolkit[2] to perform text preprocessing, including tokenization (breaking up text into words), Part-of-Speech (POS) tagging (e.g., noun, verb, adjective), lemmatization (converting words to their base forms, e.g., "emails" and "emailing" are converted to "email"), and removing stop words (i.e., noncontent words that appear too frequently in all apps, such as "a," "the").

Vocabulary construction. To get rid of noisy content that is irrelevant to app functionalities in user reviews, we need to control the size of the vocabulary. Although app descriptions may be too short for functionality extraction, the vocabulary used in app description is more formal and more relevant to app functionality. It turns out that the app description can be a good source for constructing a vocabulary. After looking at the data, we notice that most app functionalities are in the form of single nouns (e.g., navigation), noun phrases (e.g., flight information), and verb-object phrases (verb + noun, e.g., get direction). We then aggregate all app descriptions from which we only keep the single nouns, two-gram nouns, and two-gram verb-object phrases in the vocabulary. We refer to a single word or a two-gram phrase in the vocabulary as a *functional aspect*. We also remove those aspects that are too rare, that is, appearing fewer than 10 times, and too common, that is, appearing in more than 80% of the apps, from the vocabulary. Table 1 shows a simple example to illustrate the process of vocabulary construction. After this process, "find way" will be added into the vocabulary as a functional aspect.

We believe that the constructed vocabulary is able to cover most functional aspects of apps. The user review content is purified by removing the words and phrases that are not in the vocabulary.

Frequency analysis. In this step, we perform frequency analysis on the app descriptions and user reviews, and extract the most frequently mentioned functionalities for each app. We denote the vocabulary as VO. For each aspect $w \in VO$, we adapt the commonly used term weighting scheme TF-IDF (Salton & Buckley, 1988) and calculate its weight that indicates its representativeness for app a as:

$$Weight_{w,a} = rvf_{w,a} \times \log \frac{N}{af_w}.$$

where $rvf_{w,a}$ is the number of app a's user reviews that mention aspect w. $\log \frac{N}{af_w}$ is the Inverse App Frequency, where N is the total number of apps, and af_w is the number of apps that contain aspect w. If an aspect appears in too many apps (i.e., it has a high af_w value), such as "mobile," it contains little information and has low discriminating power, resulting in a low Inverse App Frequency. If the aspect appears in all apps, that is, $af_w = N$, after taking logarithm, it takes the lowest Inverse App Frequency value 0, which means it has no discriminating power. Therefore, Inverse App Frequency can be used as an indicator of the aspect's discriminating power.

We use the number of reviews that contain the aspect instead of using the frequency of the aspect in all reviews, because we believe an aspect mentioned by 10 users is more important than an aspect mentioned 10 times by one user. We also remove some aspects which are frequently

[2]http://nlp.stanford.edu/software/corenlp.shtml

DOI: 10.1002/asi

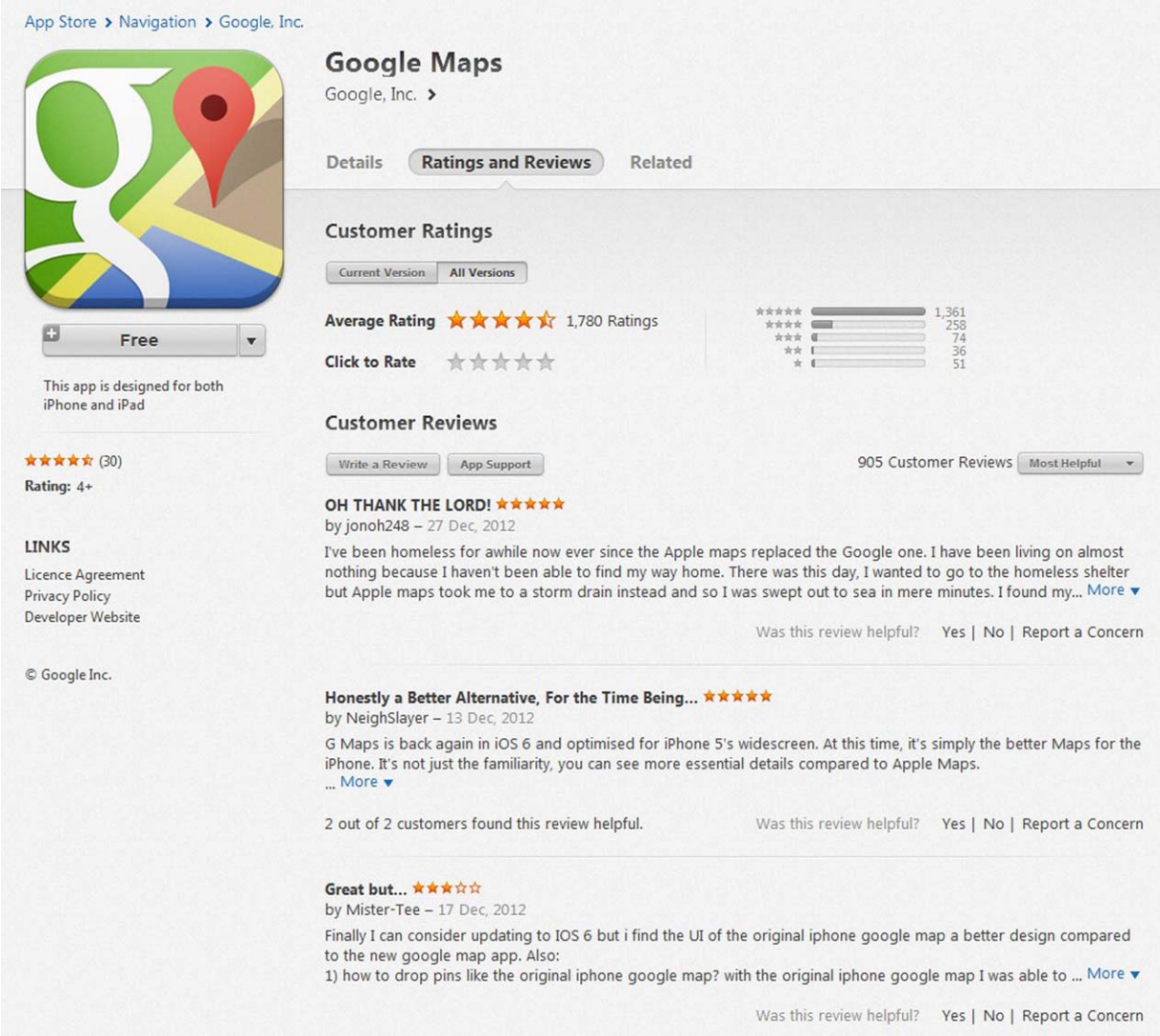

FIG. 2. User review page in apple app store. [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 1. Example of vocabulary construction.

Original sentence in the app description	POS tagging	Stop-words filtering	POS pattern filtering	Frequency filtering
You can use this app to find your way easily.	You(*pronoun*) can(*auxiliary*) use(*verb*) this(*determiner*) app(*noun*) to(*infinitive marker*) find(*verb*) your(*pronoun*) way(*noun*) easily(*adverb*)	~~You(pronoun) can(auxiliary)~~ use(verb) ~~this(determiner)~~ app(noun) ~~to(infinitive marker)~~ find(verb) ~~your(pronoun)~~ way(noun) easily(adverb)	[use app] (*verb+noun*) [find way] (*verb+noun*)	~~[use app]~~ (too frequent) **[find way]**

mentioned but not relevant to app functionalities, like app names, which are too app-specific.

After frequency analysis, we are able to obtain the functional aspects for each app by selecting the top M aspects having the highest weights.

App Recommender

Graph construction. One of the main tasks of the recommender is to predict new functionalities for the target user. We employ a graph-based ranking approach which is able to propagate users' functional requirements in the functionality graph. The first step is to construct the functionality graph that captures the cooccurrence of functionalities based on the global usage patterns from all users. Figure 3 shows an example of functionality graph.

Let $G = (V, E)$ be a directed graph with a set of vertexes V and a set of edges E. A vertex v_w denotes a functionality w, and an edge $e_{i,j}$ from vertex i to j denotes an association from functionality i to functionality j, which means if i

DOI: 10.1002/asi

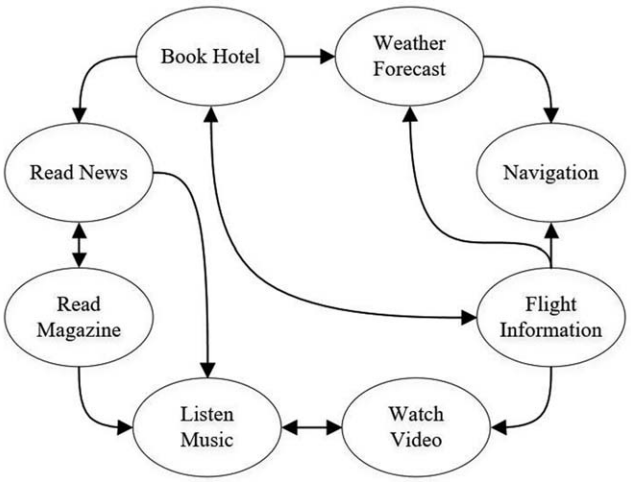

FIG. 3. Example of functionality graph.

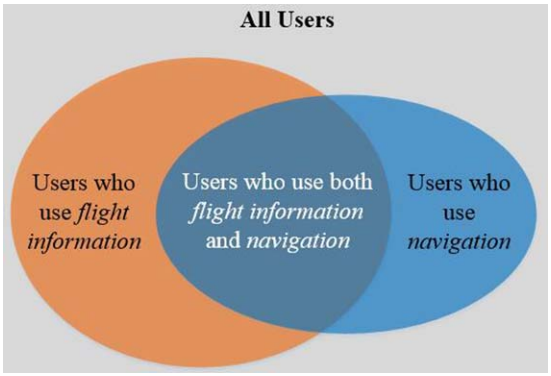

FIG. 4. Example of support and confidence. [Color figure can be viewed at wileyonlinelibrary.com]

appears, j usually appears as well. In Figure 3, for example, the edge from *navigation* to *weather forecast* indicates users who use the functionality of navigation may also use the functionality of weather forecast. We use a directed graph instead of an undirected one because the association between two functionalities is asymmetric. For example, users who use navigation may need weather forecast, but users who use weather forecast may not need navigation.

We use the well-known constraints in association rule mining, that is, *support* and *confidence*, to determine whether to add an edge into the graph. Support is a measure of usefulness of the association. An association having too low support may happen just by chance. In our context, support of an association $i \Rightarrow j$ is defined as:

$$Support(i \Rightarrow j) = \frac{|U(i,j)|}{|U|}.$$

where $|U(i,j)|$ is the number of users who install apps with functionality i and apps with functionality j, $|U|$ is the total number of users. Because the functionalities of the same app are usually homogeneous, their association cannot truly reflect the functionality usage patterns of the users. Therefore, we are interested in the association of functionalities in different apps but not in the same app. If a user installs only one app with both functionality i and j, he will not be included in $U(i,j)$. As shown in Figure 4, if we use the rectangle to denote the number of all users, and use the two ovals to denote the number of users who use *flight information* and who use *navigation* respectively, then the support of the association *flight information*⇒*navigation* can be represented as the common area of the two ovals divided by the area of the rectangle. A support value of 0.1 means that 10% of the users have both *flight information* apps and *navigation* apps on their mobile devices.

Confidence is a measure of certainty of the association. Confidence of the association $i \Rightarrow j$ can be regarded as the conditional probability of $P(j \mid i)$. In our context, it is defined as:

$$Confidence(i \Rightarrow j) = \frac{|U(i,j)|}{|U(i)|}.$$

where $|U(i,j)|$ is the number of users who install apps with functionality i and apps with functionality j, $|U(i)|$ is the number of users who install apps with functionality i. In Figure 4, the confidence of the association *flight information* ⇒ *navigation* can be represented as the common area of the two ovals divided by the area of the left oval. A confidence value of 0.4 means that among the users who have *flight information* apps on their mobile devices, 40% of them also have *navigation* apps on their mobile devices.

An edge $e_{i,j}$ is added into the graph if the association $i \Rightarrow j$ satisfies both a minimum support threshold and a minimum confidence threshold. To simplify the configuration of the model, the thresholds are usually predefined based on domain knowledge (Davale & Shende, 2015; Kaur & Madan, 2015). After referring to other studies (D. R. Liu & Shih, 2005; Sandvig, Mobasher, & Burke, 2007; Smyth et al., 2005) and consulting the industrial domain experts from Mobilewalla,[3] in our implementation, we set the minimum support threshold and the minimum confidence threshold to 0.1 and 0.4, respectively, which can effectively filter out the meaningless associations without harming the recommendation coverage. With such settings, for example, if more than 10% of users have both *flight information* apps and *navigation* apps on their mobile devices, and among the users who use *flight information* apps, more than 40% of them also use *navigation* apps, then a directed edge from vertex *flight information* to vertex *navigation* is added into the graph.

Functionality prediction. With the constructed functionality graph, we can make predictions of new functionalities for a given user. Similar to Hotho et al. (2006), we follow a two-stage random walk process to propagate user requirements from existing functionalities being used to new functionalities. At the first stage, we run the original PageRank random walk model on the functionality graph. Let $In(v_j)$ be

[3]https://www.mobilewalla.com

DOI: 10.1002/asi

the set of vertexes pointing to v_j, and Out(v_i) denote the set of vertexes pointed by v_i. The score of each vertex j at the first stage is given by:

$$PR(v_j)=(1-d)\times p(v_j)+d\times \sum_{v_i\in In(v_j)} \frac{PR(v_i)}{|Out(v_i)|}.$$

where a user follows the association to install a functionality with probability d, and jumps to a completely new functionality with probability 1-d. In our implementation, d is set to 0.85, which is an empirical value suggested by the inventors of PageRank (Page et al., 1999) to facilitate fast convergence of the algorithm and is the most common choice of other research (Boldi, Santini, & Vigna, 2005). $p(v_j)$ indicates the user's preference for functionality v_j. At the first stage, we run the nonpersonalized PageRank, so $p(v_j)$ is set to 1 for every vertex. We iterate the computation of PR score for each vertex until it converges.

The PR scores given at the first stage indicate how often each functionality co-occurs with other functionalities. For example, a high PR score of the vertex *listen music* means music apps are usually co-installed with many other apps providing different functionalities. However, what we want to know is how the user requirements may flow from the functionalities having been used to other vertex along the edges of the graph. Therefore, at the second stage, we run the personalized PageRank, in which $p(v_j)$ in Eq. 1 is given a large value (to avoid arbitrariness, similar to Kim and El Saddik [2011], we set it as $|V|$) if the functionality v_j has been used by a user. Similarly, we iterate the computation of the personalized score $PR'(v_j)$ for each vertex v_j until it converges. Then we employ a differential approach to obtain ΔPR:

$$\Delta PR(v_j)=PR'-PR.$$

The value of ΔPR in each vertex indicates the weights propagated from the functionalities that have been used by the user. It can be regarded as a measure of how likely the user needs each new functionality. For example, as shown in Figure 5, suppose a user has already used a *weather forecast* app and a *flight information app* at this stage, then the vertex *weather forecast* and the vertex *flight information* will be given a large value. Because there are many paths from *weather forecast* and *flight information* to the new vertex *navigation* that have not been used by the user, most of the weight on *weather forecast* and *flight information* will be propagated to *navigation*, reflected as a large $\Delta PR(nagivation)$. So, we can infer that this user may also need a *navigation* app. In next section, we will introduce how to utilize ΔPR for candidate set ranking.

Candidate set ranking. With ΔPR, we are able to predict new functionalities for a given user, and then we can retrieve candidate apps that contain these new functionalities. However, the candidate set generated in this way may contain

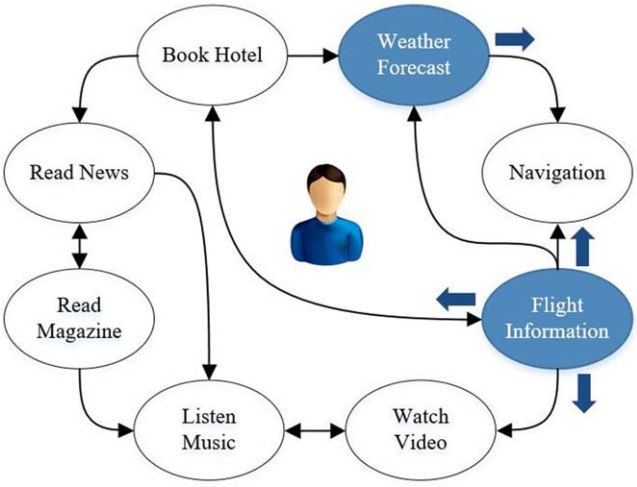

FIG. 5. Example of requirement propagation. [Color figure can be viewed at wileyonlinelibrary.com]

many apps with overlapping functionalities. To avoid redundant recommendations, we need to rank apps from the candidate set, with the following two objectives: (a) to promote apps with better quality and (b) to promote apps providing more functionalities needed by the user.

To achieve the objectives, we come up with a competition mechanism to distribute ΔPR of all new functionalities to the apps that provide these functionalities. First, for each functionality, we search for all apps that provide this functionality. Second, we rank these apps based on the average ratings, and only the one that has the highest ranking can be awarded the ΔPR of the functionality. Here our assumption is, if two apps provide similar functionalities, the one rated higher usually has better quality. Finally, for each app, we aggregate the ΔPR it wins from all functionalities it provides, to obtain the *AppRank* Score, that is:

$$AppRank(App_a)=\sum_{v_i\in Win_a} \Delta PR(v_i).$$

where Win_a is the set of functionalities for which App_a ranks higher than other apps.

We select the top N apps that have the highest *AppRank* scores as recommendations. Our mechanism allows only one app to obtain the ΔPR for each new functionality, therefore avoiding redundant recommendations. The *AppRank* score uses the summation of ΔPR from all the new functionalities, which gives priority to the apps that provide more needed functionalities.

For example, in Figure 6, suppose the app *YouTube* ranks first among both music apps and video apps; then it will be awarded the ΔPR from vertex *listen music* and *watch video*, that is, $AppRank(YouTube)=\Delta PR(listen\ music)+\Delta PR(watch\ video)=50+25=75$. Compared with other music or video apps, *YouTube* will be given a higher priority in recommendation, because it can cover two functional requirements of the user and also has good quality. Meanwhile,

DOI: 10.1002/asi

FIG. 6. Example of candidate set ranking. [Color figure can be viewed at wileyonlinelibrary.com]

other music and video apps are less likely to be recommended in this round, unless they provide additional functionalities, so redundancy is eliminated. Similarly, *Google Maps* will be awarded the ΔPR from vertex *Navigation*, that is, $AppRank(Google\ Maps) = \Delta PR(navigation) = 150$. Therefore, in this round, *Google Maps* will be the first app to be recommended, and YouTube will be the second one.

Experiment and Results

In this section, we describe the experiment we conducted to evaluate the effectiveness of our proposed solution. First, we introduce the evaluation metrics used in the experiment. Then we describe the experiment setup. Finally, we report the results, including functionality extraction, impact of sparsity, ranking accuracy, and recommendation diversity.

Evaluation Metrics

To evaluate our proposed method, we compare our method with other commonly used recommendation techniques, that is, Collaborative Filtering, Matrix Factorization, and Content-Based approach, on several evaluation metrics. A basic assumption of RSs is that "a system that provides more accurate predictions will be preferred by the user" (Shani & Gunawardana, 2011). In line with this, we evaluate our method on its prediction power. Because in many cases RSs present a list of recommendations to the user, we care more about the correctness of the predicted item ranking, which is known as ranking accuracy. However, it is widely agreed that accurate predictions are crucial but insufficient for a good RS. Users may also need to rapidly discover diverse items. Therefore, in addition to ranking accuracy, we also evaluate the diversity of recommendations.

Specifically, we use two metrics to measure ranking accuracy. The first one is Recall@k, which is defined as:

$$Recall@k = \frac{\text{num of liked items in top } k \text{ recommendations}}{\text{num of liked items}}.$$

For ranking accuracy, recall is usually measured with another metric - precision, which indicates what proportion of recommended items are liked by the users. However, because most items are unrated, it is hard to say whether the users dislike the unrated items, or they just do not know these items. Therefore, we only use the recall, which we think is more pertinent, because it only considers the liked items.

In addition to Recall@k that measures the ranking accuracy for the top N recommendations without considering the position of the N recommended items, we also use another measure, Normalized Discounted Cumulative Gain (NDCG; Herlocker, Konstan, Terveen, & Riedl, 2004), to evaluate the overall ranking accuracy, where positions are discounted logarithmically. NDCG is defined as:

$$NDCG = \frac{1}{|U|}\sum_{u\in U} Z_u \sum_{p=1}^{m} \frac{2^{r_{u,p}}-1}{\log(1+p)}$$

where U is the set of users; Z_u is a normalization factor to guarantee that for perfect ranking the NDCG value is 1; p is the position of the recommended item in the list; m is the size of candidate items; and $r_{u,p}$ is the rating given by the user u to the item at position p.

For recommendation diversity, we use 1 - Intra-List Similarity (Järvelin & Kekäläinen, 2002), which is defined as:

$$Diversity = 1 - ILS = 1 - \frac{\sum_{Rec_i\in Rec}\sum_{r_a\in Rec_i}\sum_{r_b\in Rec_i, r_b\neq r_a} Sim(r_a, r_b)}{2|Rec|}.$$

where Rec is the set of recommended items for all users; Rec_i is the list of recommended items for user i; r_a and r_b are two different items in user i's recommendation list; and $Sim(r_a, r_b)$ measures the content similarity between item r_a and r_b, which is the proportion of overlapping functional aspects of two apps in our implementation:

$$Sim(r_a, r_b) = \frac{|W_a \cap W_b|}{|W_a \cup W_b|}.$$

where W_a and W_b are the sets of functional aspects of recommended app a and app b, respectively.

Experiment Setup

The data we use in the experiment is obtained from the Apple App Store (U.S.).[4] We construct the vocabulary based on the textual descriptions of 10,530 popular apps evenly distributed in 22 categories. The constructed vocabulary contains 20,690 words and phrases. Our constructed dataset

[4]https://itunes.apple.com/us/genre/mobile-software-applications/id36?mt=8

DOI: 10.1002/asi

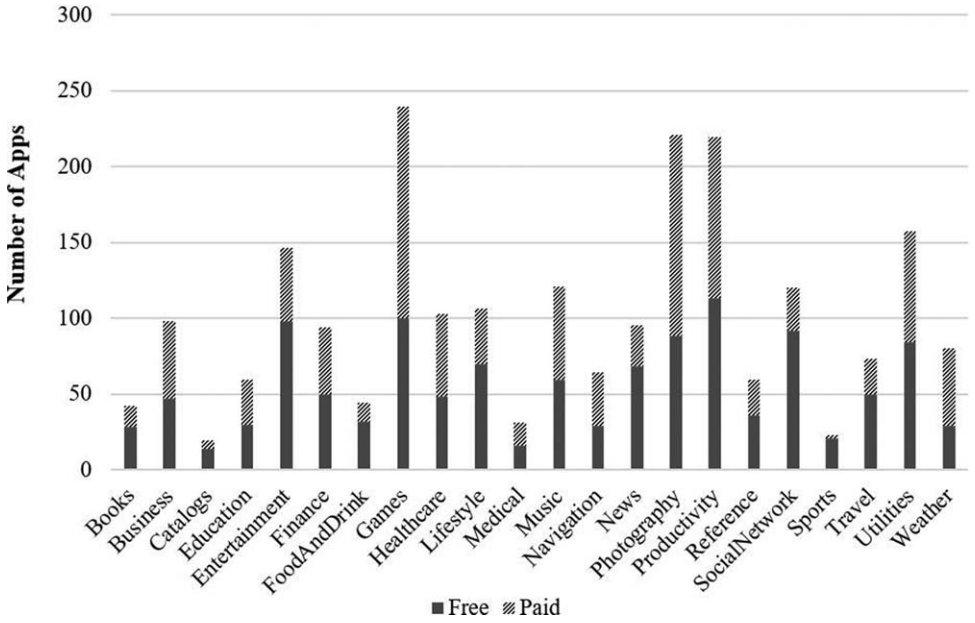

FIG. 7. App category distribution.

for evaluation contains 66,543 ratings on a scale of 1 to 5 given by 1,879 users to 2,213 apps. The sparsity level (i.e., the percentage of empty entries in the user-app rating matrix) of the data set is 98.39%. In the dataset 1,202 apps are free, and the remaining 1,101 apps are paid. The distribution of app categories in our dataset is shown in Figure 7. In the dataset, each user has rated at least 5 free apps and 5 paid apps. On average, each user has rated 20 free apps and 15 paid apps. For each app in the dataset, we collected a maximum of 500 user reviews. On average, each app had 442 reviews.

To test the impact of the number of extracted aspects M on the model, we vary its value and then compare the recommendation quality metrics mentioned above, as well as two other system performance measures: the coverage (measured by the percentage of apps that can be recommended) and the training time of the model with 80% training data.

The results in Figure 8 show that the coverage of recommendation increases when a larger number of M are used. Higher coverage is preferred in recommendation because it indicates the ability of the recommender system to explore new items. However, the results in Figure 8 also show that the training time of the model moves up when M increases, and there is a rapid rise of training time when M increases from 50 to 60. To make a trade-off between the coverage and the efficiency of the model, M is fixed to 50, where the training time is relatively low, and the coverage level (97.8%) is acceptable.

We find that the number of extracted aspects M shows no significant effect on the recommendation quality. One possible explanation is that the two-stage PageRank process may filter out those irrelevant aspects even if they are added into the graph, so the ranking accuracy is stable using a different number of aspects.

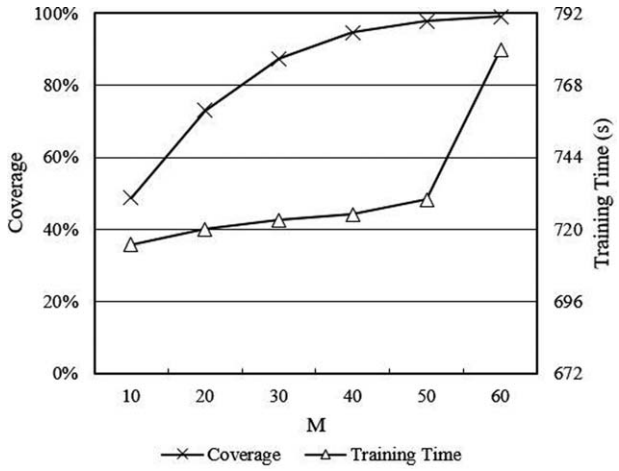

FIG. 8. Impact of M.

Experiment Results

Qualitative results for functionality extraction. To investigate the effectiveness of our method for extracting app functionalities, we select 5 popular apps and for each app, we list only the top 12 extracted functionalities using our method. The qualitative results are shown in Table 2.

From the results, we can see that most of the extracted functionalities are meaningful and reasonable. The quality of the extracted functionalities plays an important role in the whole solution, because the functionalities are the basis of further analysis for recommendation. The results show that *our proposed method is effective in extracting app functionalities of good quality from user reviews, which guarantees the effectiveness of the whole solution.*

DOI: 10.1002/asi

TABLE 2. Extracted functionalities.

App name	Functionalities
Dropbox	doc, file, space, photo, video, computer iphone, access file, share link, access photo, video device, share photo, attachment
WhatsApp	message, messager, chat, group, contact, friend, address book, chat history, friend world, send message, group chat, voice note
Kindle	book, newspaper, textbook, magazine, reader, bookmark, reading, reading experience, book mark, read book, pdf, dictionary
Gmail	mail, google, conversation, inbox, receive email, account support, attachment, get notification, account, mail app, contact, send email
YouTube	video, playlist, google, video playlist, list search, watch video, watch list, share video, channel, search video, share friend, entertainment

Impact of sparsity. In this experiment, we compare our method with other methods for generating the top N recommendations using different training-test ratios. The methods we compare with are commonly selected for comparison in recommendation research: User-Based CF (UCF; Resnick et al., 1994), Item-Based CF (ICF; Sarwar et al., 2001), Content-Based Filtering (CB; Pazzani & Billsus, 2007), Non-Negative Matrix Factorization (NMF; Lee & Seung, 1999), Regularized Singular Value Decomposition (RSVD) and its variant SVD++ (Paterek, 2007). For the CB method, the inputs are user ratings and extracted functionalities, which are the same as our proposed method. The inputs of other methods are only user ratings.

We introduce a variable *tp* to indicate what percentage of the rating data is used as the test set. For example, $tp = 10\%$ indicates 10% of the data is used as the test set, and the remaining 90% of the data is used as the training set. A rating in the test set is converted into "like" if its value is larger than 3. We fix $N = 100$, and vary the percentage of the test data $tp = 10\%, 20\%, \ldots, 90\%$. The corresponding recall values are shown in Figure 9.

The results show that our proposed *AppRank* method is less sensitive to the training-test ratio compared to other methods, and it always outperforms other methods on all *tp* values. As *tp* increases, less data is used for training, which means the sparsity level of the training set increases as well. Therefore, the results also show that our method is less sensitive to data sparsity, and its improvement is more salient in extremely sparse settings. Specifically, when $tp = 90\%$, our *AppRank* method increases the recall of the second best method, that is, CB, from 0.12 to 0.23. The results show *the effectiveness of our AppRank method in alleviating data sparsity*.

Comparison of top N recommendations. In this experiment, we fix the *tp* to 60%, where most methods have high recall, and vary the number of recommended apps $N = 10, 20, \ldots, 100$ to compare the recall of different methods for the top N recommendations. The results of the comparison in Figure 10 show that the recalls of all methods increase along with the N, and the recall of our method outperforms all other methods for different N. The results indicate that *our AppRank method has significant improvement on ranking accuracy for the top N recommendations*.

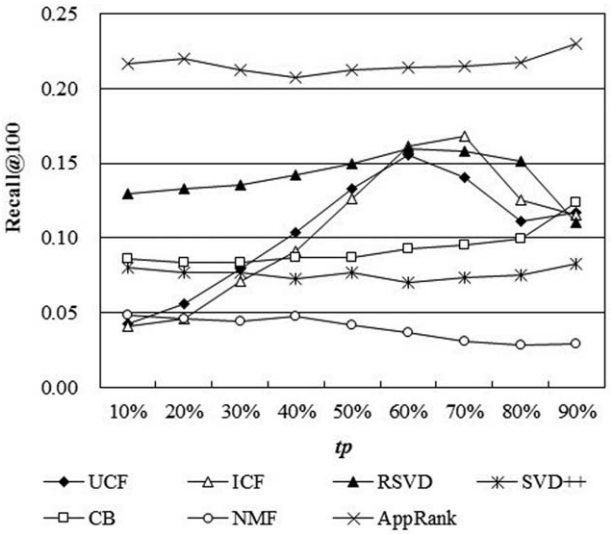

FIG. 9. Comparison of recall@100 with different TP. Higher recall indicates higher accuracy.

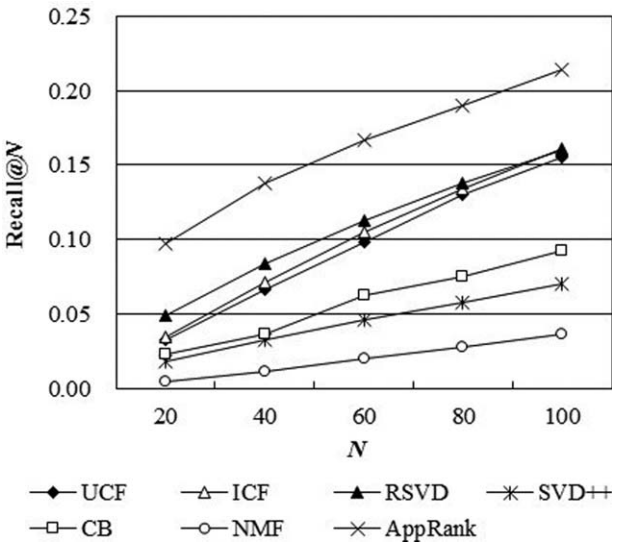

FIG. 10. Comparison of recall@N. Higher recall indicates higher accuracy.

Comparison of overall ranking. In this experiment, we fix the *tp* to 60% and compare the NDCG values of different methods to investigate the correctness of overall rankings for all candidate items. From the results shown in Figure 11, we can see that our proposed *AppRank* method has the highest NDCG value, and it increases the NDCG value of the second-best method, that is, RSVD, by 14.27%. The results

DOI: 10.1002/asi

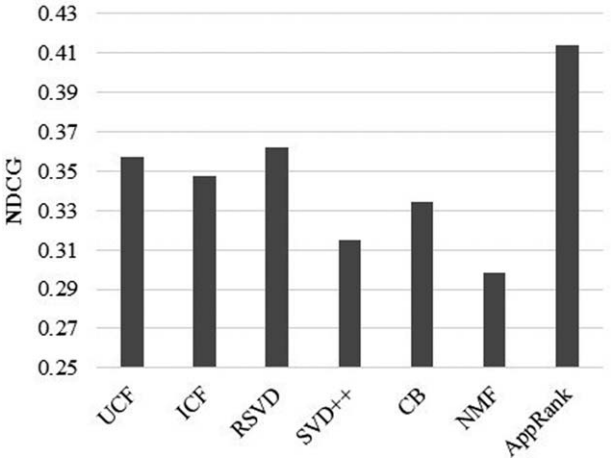

FIG. 11. Comparison of NDCG. Higher NDCG indicates higher accuracy.

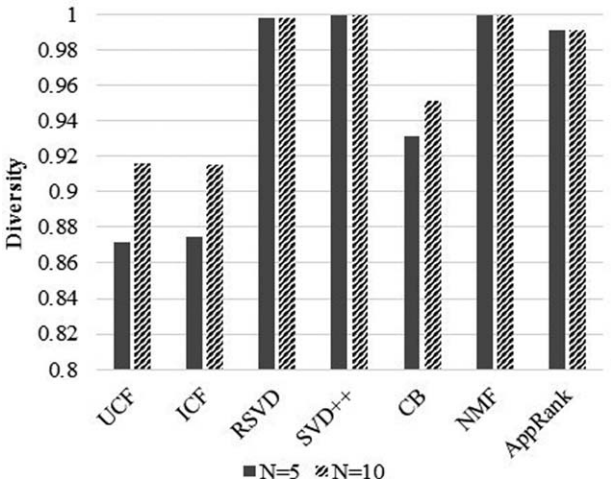

FIG. 12. Comparison of diversity. Higher diversity indicates better result.

show that *our method is effective in improving the correctness of the overall ranking for all candidate apps.*

Comparison of recommendation diversity. We find that at less sparse settings, generally all methods are able to generate diverse recommendations. However, when the training data becomes sparse, the diversity of some methods drops down. We set $tp = 90\%$, and compare the diversity of the top 5 and top 10 recommended apps of different methods. The results are shown in Figure 12. From the results, we can see that the diversity of the top 5 and top 10 recommended apps generated by our method remains high, which is 0.9913 and 0.9916 respectively. However, for UCF, ICF, and CB, the diversity is significantly lower. For instance, the diversity of the top 5 and top 10 recommended apps generated by UCF is only 0.8715 and 0.9164, respectively. The results indicate that *our method is less sensitive to data sparsity in terms of recommendation diversity.*

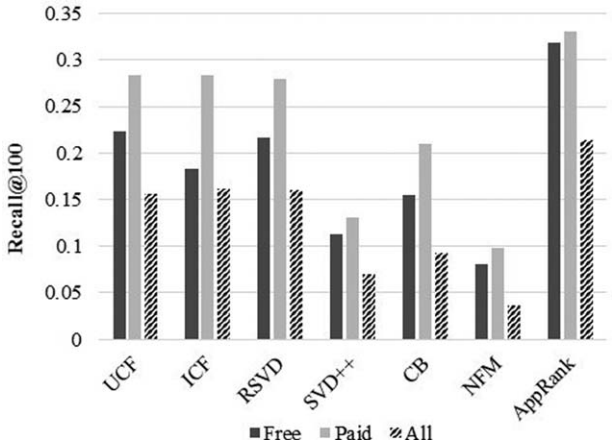

FIG. 13. Comparison of recall@100 for free and paid subsets. Higher diversity indicates better result.

Comparison between free and paid apps. In this experiment, we split the dataset into two subsets. One subset contains only free apps and another contains only paid apps. We set $tp = 60\%$ and compare Recall@100 of different methods on these two subsets as well as the whole dataset respectively. The results are shown in Figure 13. From the results, we find that for all methods, the recall values for both free app and paid app subsets are higher than for the whole dataset. This implies *that users' interests and requirements are easier to predict within one class (either free apps or paid apps).* Moreover, the *recall values for the paid app subset are higher than for the free app subset.* This is reasonable, because users will consider more about what they need when they are installing paid apps so it is easier to capture their requirements. On either the free app or paid app subset, *our proposed AppRank method outperforms all the other methods.*

Conclusion

In this article, we propose a functionality-based mobile app recommendation architecture. Our method recommends apps by revealing their detailed functionalities of apps and truly capturing users' functional requirements, which have not been considered by existing works. Furthermore, we show that user reviews can be used to enrich item information and can be incorporated to enhance recommendations. The experiment conducted on a real-world dataset shows that our proposed *AppRank* method is robust in data sparse settings, and it is able to significantly improve recommendation accuracy and diversity. In particular, our proposed *AppRank* method doubles the recall value of the second best method (CB) under an extremely sparse setting, increases the overall ranking accuracy of the second best method (RSVD) by 14.27%, and retains a high diversity of 0.99.

This research not only provides theoretical contributions to recommendation literature, but also has practical implications. The proposed solution can be implemented as an effective real-world app RS helping users to discover apps

DOI: 10.1002/asi

that meet their requirements. The recommended apps would be more accurate, more diverse, and have fewer overlapping functionalities.

Our solution has some limitations. First, when ranking the candidate apps with similar functionalities, we simply use the average rating. In future work, it would be possible to extract other non-functional aspects from user reviews, which can be incorporated in the ranking process to enable a sophisticated ranking approach. Second, as the rating data were collected from active users in the evaluation, it may have some selection bias. This can be addressed in future work by using a more general dataset and conducting user studies. Third, because most apps in the current dataset have sufficient reviewers, our method can be extended by incorporating other information to consider the situation where some apps have fewer reviewers. In addition, our method focuses more on the apps providing functionalities for users. However, there are also apps that may not be functionality-oriented. In future work, we will investigate the impact of product category on user requirement modeling and extend our work by coming up with strategies to capture user requirements by differentiating utilitarian and hedonic products in recommendation.

Acknowledgments

We gratefully acknowledge the funding support from the National Natural Science Foundation of China (Grant 71601081 and 71503084), the Guangdong Natural Science Foundation (Grant 2016A030310426) and the Fundamental Research Funds for the Central Universities (Grant 2017BQ048 and 2017BQ049). Dr. Chunmian Ge is the corresponding author.

References

Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering, 17, 734–749.

Böhmer, M., Bauer, G., & Krüger, A. (2010, September). Exploring the design space of context-aware recommender systems that suggest mobile applications. Paper presented at the 2nd Workshop on Context-Aware Recommender Systems, CARS 2010, Barcelona, Spain.

Boldi, P., Santini, M., & Vigna, S. (2005, May). PageRank as a function of the damping factor. Paper presented at the International Conference on World Wide Web. Chiba, Japan: ACM.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. User Modeling and User-Adapted Interaction, 12, 331–370.

Davale, A.A., & Shende, S.W. (2015, February). Implementation of coherent rule mining algorithm for association rule mining. Paper presented at the 2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), ABLAZE 2015, Noida, India: IEEE.

Felfernig, A., Mandl, M., Schippel, S., Schubert, M., & Teppan, E. (2010). Adaptive utility-based recommendation. Berlin: Springer.

Guan, Z., Bu, J., Mei, Q., Chen, C., & Wang, C. (2009, July). Personalized tag recommendation using graph-based ranking on multi-type interrelated objects. Paper presented at the International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009, Boston, MA.

Herlocker, J.L., Konstan, J.A., Terveen, L.G., & Riedl, J.T. (2004). Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems (TOIS), 22, 5–53.

Hosanagar, K., Fleder, D., Lee, D., & Buja, A. (2013). Will the global village fracture into tribes? Recommender systems and their effects on consumer fragmentation. Management Science, 60, 805–823.

Hotho, A., Jäschke, R., Schmitz, C., Stumme, G., & Althoff, K.-D. (2006). Folkrank: A ranking algorithm for folksonomies. Paper presented at the LWA Conference, LWA 2006, Hildesheim, Germany.

Huang, Z. (2004). Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. ACM Transactions on Information Systems, 22, 116–142.

Järvelin, K., & Kekäläinen, J. (2002). Cumulated gain-based evaluation of IR techniques. ACM Transactions on Information Systems (TOIS), 20, 422–446.

Jeh, G., & Widom, J. (2003, May). Scaling personalized web search. Paper presented at the 12th International Conference on World Wide Web. Budapest, Hungary: ACM.

Karatzoglou, A., Baltrunas, L., Church, K., & Böhmer, M. (2012, October–November). Climbing the app wall: Enabling mobile app discovery through context-aware recommendations. Paper presented at the 21st ACM International Conference on Information and Knowledge Management. Maui, HI: ACM.

Kaur, J., & Madan, N. (2015). Association rule mining: A survey. International Journal of Hybrid Information Technology, 8, 239–242.

Kim, H.N., & El Saddik, A. (2011, October). Personalized PageRank vectors for tag recommendations: Inside FolkRank. Paper presented at the ACM Conference on Recommender Systems, Recsys 2011, Chicago, IL.

Lee, D.D., & Seung, H.S. (1999). Learning the parts of objects by non-negative matrix factorization. Nature, 401, 788–791.

Lin, J., Sugiyama, K., Kan, M.-Y., & Chua, T.-S. (2013, July–August). Addressing cold-start in app recommendation: Latent user models constructed from twitter followers. Paper presented at the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval. Dublin, Ireland: ACM.

Liu, B., Kong, D., Cen, L., Gong, N.Z., Jin, H., & Xiong, H. (2015, January–February). Personalized mobile app recommendation: Reconciling app functionality and user privacy preference. Paper presented at the 8th ACM International Conference on Web Search and Data Mining. Shanghai, China: ACM.

Liu, D.R., & Shih, Y.Y. (2005). Integrating AHP and data mining for product recommendation based on customer lifetime value. Information & Management, 42, 387–400.

Liu, Q., Ma, H., Chen, E., & Xiong, H. (2013). A survey of context-aware mobile recommendations. International Journal of Information Technology & Decision Making, 12, 139–172.

Mihalcea, R., & Tarau, P. (2004). TextRank: Bringing order into texts. Paper presented at the Conference on Empirical Methods in Natural Language Processing.

Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The PageRank citation ranking: Bringing order to the web. Stanford Digital Library Technologies Project.

Paterek, A. (2007, August). Improving regularized singular value decomposition for collaborative filtering. Paper presented at the KDD Cup and Workshop. San Jose, CA: ACM.

Pazzani, M.J., & Billsus, D. (2007). Content-based recommendation systems. In The adaptive web (pp. 325–341). Berlin: Springer.

Pu, P., Faltings, B., Chen, L., Zhang, J., & Viappiani, P. (2011). Usability guidelines for product recommenders based on example critiquing research. In Recommender systems handbook (pp. 511–545). New York: Springer.

Resnick, P., Iacovou, N., Suchak, M., Bergstorm, P., & Riedl, J. (1994, October). GroupLens: An open architecture for collaborative filtering of netnews. Paper presented at the ACM Conference on Computer Supported Cooperative Work. Chapel Hill, NC: ACM.

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. Information Processing & Management, 24, 513–523.

Sandvig, J.J., Mobasher, B., & Burke, R. (2007). Robustness of collaborative recommendation based on association rule mining. Paper presented at the Proceedings of the 2007 ACM Conference on Recommender Systems, Minneapolis, MN.

Sarwar, B., Karypis, G., Konstan, J., & Reidl, J. (2001, May). Item-based collaborative filtering recommendation algorithms. Paper presented at the 10th International Conference on World Wide Web. Hong Kong: ACM.

Shani, G., & Gunawardana, A. (2011). Evaluating recommendation systems. In F. Ricci, L. Rokach, & B. Shapira (Eds.), Recommender systems handbook (pp. 73–105). New York: Springer.

Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., Hanjalic, A., & Oliver, N. (2012). TFMAP: Optimizing MAP for top-n context-aware recommendation. Paper presented at the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval.

Smyth, B., McCarthy, K., Reilly, J., O'sullivan, D., McGinty, L., & Wilson, D.C. (2005, June). Case studies in association rule mining for recommender systems. Paper presented at the IC-AI 2005 International Conference on Artificial Intelligence, ICAI 2005, Las Vegas, NV.

Spiekermann, S., & Paraschiv, C. (2002). Motivating human–agent interaction: Transferring insights from behavioral marketing to interface design. Electronic Commerce Research, 2, 255–285.

Xu, X., Dutta, K., & Datta, A. (2014). Functionality-based mobile app recommendation by identifying aspects from user reviews. Paper presented at the 35th International Conference on Information Systems (ICIS), Auckland, New Zealand.

Yan, B., & Chen, G. (2011, June–July). AppJoy: personalized mobile application discovery. Paper presented at the 9th International Conference on Mobile Systems, Applications, and Services, Washington, DC.

Yi, M., & Deng, W. (2009, July). A utility-based recommendation approach for E-Commerce websites based on Bayesian networks. Paper presented at the Second International Conference on Business Intelligence and Financial Engineering, Bife 2009, Beijing, China.

DOI: 10.1002/asi