Functionality-Based Mobile App Recommendation by Identifying Aspects from User Reviews

Research-in-Progress

Xiaoying Xu
Department of Information Systems
National University of Singapore
15 Computing Drive, Singapore, 117418
xu.xiaoying@comp.nus.edu.sg

Kaushik Dutta
Department of Information Systems
National University of Singapore
15 Computing Drive, Singapore, 117418
duttak@nus.edu.sg

Anindya Datta
Department of Information Systems
National University of Singapore
15 Computing Drive, Singapore, 117418
datta@comp.nus.edu.sg

Abstract

The explosive growth of mobile apps makes it difficult for users to find their needed apps in a crowded market. 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 app domain. In this paper, we propose a recommendation architecture that is able to generate app recommendations at functionality level. We address the redundant recommendation problem in app domain by highlighting users’ functional requirements that have received scant attention from existing recommendation research. Another main feature of our work is extracting app functionalities from textual user reviews for recommendation. Effective approach for functionality extraction is also proposed.

Keywords: Mobile app, recommender system, text mining, user review
Introduction

Accelerated by the popularity of smart phones, mobile application (or app for short) market is growing explosively. For instance, Apple App Store provides more than one million apps in 24 categories for users in 155 countries around the world\(^1\). On the one hand, tens of thousands of new apps are continuously released in app stores, but most of them can hardly get reached by users via keyword search; on the other hand, it has been a significant challenge for users to find their needed apps in such crowded app stores. Therefore, effect mechanisms to help users discover new apps among overwhelming number of alternatives are very necessary.

To alleviate the new item discovery problem, many industry solutions, i.e. personalized recommender systems (RS), in other consumer product domains, e.g. book, movie, music etc., have been proposed. These solutions mostly deal with new item problem by recommending items that are similar to those the user has selected (Celma et al. 2005; Rafailidis et al. 2014; Schwab et al. 2001; Semeraro et al. 2009). While the general goal of mobile app recommendation is similar to those in traditional domains – to guide users to items that are relevant to their interests, there are unique features of mobile app that make the solutions in traditional domains less effective in app domain.

One of the most important characteristics of mobile app is that the basis for users to select apps is more of the apps’ functionalities than the users’ taste. 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, e.g. whether forecast, he/she needs no more similar apps with the same functionality of whether forecast, unless they provide additional functionalities. If existing recommendation techniques are directly applied in app domain, users may be end up with receiving a mass of redundant app recommendations providing similar functionalities.

Moreover, the most widely used recommendation techniques, i.e. Collaborative Filtering (CF) (Sarwar et al. 2001) and Content-based Filtering (CBF) (Pazzani and Billsus 2007), usually generate recommendations based on user ratings. In app domain however, rating values indicate more about users’ evaluation on the non-functional aspects (ease of use, UI design, power consumption etc.) of the app, but can hardly reflect users’ functional requirements. For example, even if a user gives a very low rating to an app providing weather forecast, we can only say the user is not satisfied with this app (maybe because it is power consuming), but we cannot deny the fact that this user needs the functionality of weather forecast, since he has been attracted by the described functionalities of this app and has decided to install it. Therefore, when applied in app domain, traditional techniques fail to reveal the detail functionalities inside apps, and lack the ability to capture users’ functional requirements, which may worsen the quality of recommendations.

Recently, an increasing amount of research has paid attention to mobile app recommendation. These works have enjoyed varying degrees of success by either adapting traditional recommendation techniques to app domain (Bhandari et al. 2013; Lin et al. 2013; Yan and Chen 2011), or considering additional dimension of app (e.g. context information) (Böhmer et al. 2010; Karatzoglou et al. 2012; Shi et al. 2012). However, the redundancy problem in app recommendation has received scant attention from researchers, and no reported work has been found that recommends apps by considering user requirements at the functionality level.

To bridge this gap, in this paper, we aim at proposing a functionality-based recommendation architecture that is able to provide more accurate and more diverse app recommendations by analyzing users’ functional requirements. In our proposed solution, a mobile app is modelled as a collection of different functionalities, and user requirements are modelled at the functionality level. We first predict what new functionalities a given user most likely needs based on other users’ usage patterns, and select a collection of apps containing these new functionalities as recommendations. If there are similar apps providing overlap functionalities in this collection, we then estimate the user’s preference towards each similar app based on other non-functional aspects captured by user ratings, and only recommend the one that the user most likely prefers, therefore truly capturing users’ functional requirements and avoiding redundant recommendations.

---

We achieve our goal by solving three important problems. First, given an app, we need to know what functionalities it has provided. Although some functionalities are explicitly stated in the apps' descriptions, they are embedded in short text blocks and hard to be extracted using the descriptions alone. We note however, that the functionalities of an app may be repeatedly mentioned in the app's user reviews. In addition, user reviews may also contain other implicit functional aspects that are not stated in the descriptions but are useful for modeling user requirements. Therefore, one main feature of our solution is to obtain functionalities of apps by mining textural user reviews. To accurately extract both explicit and implicit functional aspects of apps from noisy review content, we propose a simple but effective approach by conducting within-app and cross-app analysis.

Second, user requirements should be properly modelled. We aggregate the functionalities of all apps installed by each user and represent user requirements as functionality vectors. We also propose functionality-based CF to predict new functionalities that may be needed by the user.

Third, we need to rank and select good apps from similar candidates providing overlap functionalities to avoid redundancy. Noticing that users would consider other non-functional aspects when they are comparing apps among similar alternatives, we propose to map users' non-functional requirements into latent dimensions by applying Singular Value Decomposition (SVD) (Paterek 2007) to user ratings. The ranking of similar apps is given by matching up apps' non-functional aspects with users' non-functional requirements in latent space.

The remaining of the paper is organized as follows: first we review existing works on mobile app recommendation in literature. Then we describe the intuition and overview followed by the details of our proposed architecture. The remainder of the paper then presents the results of our preliminary evaluation. Finally, we discuss the expected contribution and future work.

**Mobile App Recommendation**

Recently, researchers start paying attention to mobile app recommendation, and an increasing amount of research on app recommendation has been found. A few studies propose to extend traditional recommendation algorithms and to adapt them into app domain. For example, *AppJoy* (Yan and Chen 2011) replaces the user ratings in traditional RS with usage scores composed by recency, frequency and duration, and then performs item-based CF recommendation. Bhandari et al. (2013) adapt graph-based recommendation for app discovery, aiming at improving novelty. Lin et al. (2013) propose to extend model-based RS by constructing latent user models from apps’ twitter followers, addressing cold-start problem of app recommendation. Hybrid methods are also existing. For example, Xia et al. (2014) report a multi-object approach to evolve existing mobile app RSs. Although these solutions have been proved to be effective to some extent in recommending apps, they do not consider much about app’s unique characteristics.

Noticing this limitation, some researchers have shifted their focus to a unique characteristic of mobile app – context, and a few context-aware systems have been proposed in app domain. Such systems record users' context information, e.g. physical location, at a particular time and then enhance app recommendation by exploiting the collected context information (Liu et al. 2013). For example, Böhm et al. (2010) explore the design space for context-aware app recommendation, and develop a prototype app RS in Android platform called *Appazaar*. Djinn model introduced by Karatzoglou et al. (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 to generate app recommendations by optimizing the ranking (i.e. MAP). Context-aware app RSs are highlighted since they take into account one important feature of mobile app, i.e. context information. Such systems show better performance than traditional methods in recommending apps. However, context information is very difficult to be collected due to privacy concerns and other constrains. It has been a significant limitation of context-aware systems.

To conclude, existing works on mobile app recommendation do consider some unique features in app domain; however, no reported work has been found to recommend apps at functionality level and to avoid redundant recommendations, which will be highlighted in our proposed method.
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/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 non-functional 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: (1) knowing all functionalities provided by the apps that a user has been using; (2) predicting what other functionalities this user may need; and (3) helping the user select his/her preferred apps providing these new functionalities.

For example, let's assume that the target user has installed an app providing weather forecast and airline information in his/her mobile phone. By analyzing other users’ usage patterns, we find that users who use apps providing weather forecast or airline information may also use apps providing navigation that the target user has not installed. We then select a set of apps providing navigation as recommendation candidates. Users may provide feedback by rating the apps they are using or have tried. Advanced global rating analysis allows us to know what non-functional aspects the target user cares (e.g. he/she may highly value the UI design of an app), and what non-functional features each app has. Therefore, we are able to recommend the correct navigation app having the best UI design to the target user.

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. But we quickly realize that descriptions are short text where 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 (Hu and Liu 2004). It motivates us to obtain app functionalities from user reviews. However, it is common that user reviews contain many things that are not relevant to app functionalities. In order to filter out noisy content, we propose to use apps’ description content as reference to construct vocabulary, and perform within-app and cross-app analysis to user reviews, which contributes to extracting not-too-specific and not-too-general feature words and phrases related to app functionalities. Besides, after getting app functionalities, we adopt the idea of Collaborative Filtering and propose functionality-based CF to discover new functionalities for users, taking other users’ usage patterns into consideration. We also apply SVD to user ratings to analyze their non-functional requirements and intelligently filter out apps with overlap functionalities, therefore capturing user requirements and addressing 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.

Architecture

Our proposed app recommendation architecture is shown in Figure 1. There are three main components in the architecture, they are: 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, App Recommender predicts new functionalities for the user, selects candidate apps to recommend, and intelligently filters out apps with overlap functionalities. More details of each component will be introduced in the ensuing sections of the paper.
Functionality-Based Mobile App Recommendation

App Data Crawler

The main task of the crawler is to collect web pages containing app descriptions and user reviews from app store. Since 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 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 based forms, e.g. “emails” and “emailing” are converted to “email”), and removing stop words (i.e. meaningless words that appear too frequently in all apps, like “a”, “the”). For single words, we only keep the nouns because we believe nouns are more relevant to app functionalities. We also generate 2-grams (e.g. “send email”) consisting of verbs and nouns. In the following, we refer to a single word or a 2-gram phrase as an aspect.

Within-app Analysis. This step is to find out the functional aspects that are frequently mentioned in an app’s user reviews by computing their term weights and selecting those aspects having highest weights. Since review content may be very noisy with a lot of irrelevant information, we propose to use apps’ descriptions as reference. In other words, we constrain the vocabulary to those aspects that only appear in the descriptions of apps, since descriptions are more formal and less noisy. We construct the vocabulary using a subset of diverse apps under different app categories. We also filter out those aspects that rarely appear (i.e. appear less than 5 times). We believe the constructed vocabulary is able to cover most functional aspects of apps.

We denote the vocabulary as \( V \), for each aspect \( w \in V \), we calculate its term weight that indicates its representativeness of app \( a \) by adding the review frequency \( rf \) into traditional \( tf-idf \) (Salton and McGill 1983) term weighting scheme:

\[
\text{Weight}_{w,a} = \left( \frac{tf_{w,a}}{\text{DesLength}_a} + \frac{rf_{w,a}}{\text{RevLength}_a} \right) \times idf_w = \left( \frac{tf_{w,a}}{\text{DesLength}_a} + \frac{rf_{w,a}}{\text{RevLength}_a} \right) \times \log \frac{N}{df_w}.
\] (1)

\( tf_{w,a} \) is the frequency of aspect \( w \) in app \( a \)'s description, and \( rf_{w,a} \) is the frequency of aspect \( w \) in app \( a \)'s user reviews, they are normalized by the description length \( \text{DesLength}_a \) and review length \( \text{RevLength}_a \) respectively. \( idf_w \) is the inverse document frequency that indicating the term’s discriminating power, where \( N \) is the total number of apps, and \( df_w \) is the number of apps whose description contain aspect \( w \). For example, let’s assume that the aspect “send email” appears 1 time in an app’s description with 70 words,

\(^2\) http://nlp.stanford.edu/software/corenlp.shtml
and appears 500 times in its review content with 40000 words. We use 5000 apps to construct the vocabulary, and among them 20 apps contain “send email”. Then the weight of “send email” for this app is: $\left(\frac{1}{70} + \frac{500}{40000}\right) \times \log \left(\frac{40000}{20}\right) = 0.21$.

The proposed term weighing scheme uses the summation of the normalized description term frequency $tf$ and review term frequency $rf$ that considers the situation where the app’s description is extremely short and does not contain informative content. In such case, $rf$ allows us to find out frequently mentioned functional aspects that are not explicitly stated in description (i.e. $tf$ is zero). Similarly, we also consider those apps without any user reviews by allowing $rf$ to be zero and therefore only using their description (i.e. $tf$).

**Cross-app Analysis.** After within-app analysis, a list of functional aspects with highest weights are extracted for each app. However, we notice that the functional aspects given by within-app analysis may be too specific to be used for finding other apps. For example, the word “iBook” that is the name of an Apple app *iBook* may appear in the app’s description and be frequently mentioned in its user reviews. However, this aspect is too specific since we can hardly find other reading app providing the functionality called “iBook”. Instead, the aspects “book” and “reading” are much better for this app. In order to alleviate this problem, we propose to perform cross-app analysis that uses the similar apps’ term weights to moderate the original term weights given by within-app analysis. For example, let’s assume that we find other two reading apps providing similar functionalities to *iBook* and therefore the aspects “book” and “reading” may have high weights in both of them, but they do not contain the aspect “iBook”. For each aspect, we take the average of its term weights in three apps, so the weights of “book” and “reading” still remain high, but the weight of “iBook” becomes $1/3$ of its original value since it does not appear in other two apps, therefore we lower the weights of too specific aspects.

We achieve this by using the aspects and their weights given by within-app analysis to construct aspect vectors for each app, and finding similar apps by calculating the cosine similarities between the aspect vectors:

$$\text{CosSim}(\vec{V}_1, \vec{V}_2) = \frac{\vec{V}_1 \cdot \vec{V}_2}{\|\vec{V}_1\| \|\vec{V}_2\|}.$$  \hspace{1cm} (2)

For a given app, we find $N$ similar apps for it. For each functional aspect $w$ in app $a$, the moderated weight is given by:

$$\text{Weight}'_{w,a} = \frac{\alpha \text{Weight}_{w,a} + \left(1 - \alpha\right) \sum_{j=1}^{N} \text{Weight}_{w,j} \cdot \frac{1}{\alpha + (1 - \alpha)N}}{\alpha + (1 - \alpha)N}.$$  \hspace{1cm} (3)

where $\text{Weight}_{w,a}$ is the original weight given by within-app analysis, $\text{Weight}_{w,j}$ is the weight in the $k$th similar app, and $\alpha$ is a parameter indicating the extent of moderation, $\alpha \in (0,1)$.

After within-app and cross-app analysis, we are able to obtain the functional aspects that are not-too-specific and not-too-general for each app by selecting those aspects whose moderated weights are over a threshold $\text{Weight}'$. Each extracted functional aspect indicates one functionality of the app.

**App Recommender.**

**Functionality Prediction.** The first task of recommender is to predict new functionalities for the target user. We replace item in item-based CF (Sarwar et al. 2001) by functionality and propose functionality-based CF, aiming at finding new functionalities that are relevant to other functionalities the user has been using.

Different from traditional CF, we drill down into the functionality level and use rating counts instead of rating values to construct functionality vectors, considering the fact that user’s functional requirements may be captured by the rating behavior itself other than the rating value. For example, if a user has rated an app providing weather forecast, it means this user needs the functionality of weather forecast no matter what rating he has given, since he has been attracted by the functional description of this app and has decided to install it. If this user has rated several apps providing weather forecast, it means weather forecast is so
important to this user that he/she wants to find out the best one by trying several candidates. By analyzing other users’ rating behavior, we may find that users who need weather forecast also need navigation, then we predict navigation is also an important functionality to this user.

We represent each functionality \( f \) as a vector \( V_f \) in which each element is associated with one user who has given rating to the apps providing this functionality. And the value of the element is the user’s rating of the element, indicating the extent to which the user needs this functionality. Then for each new functionality that has not been used by the target user \( u \), we measure its relevance to each of the functionalities that the target user already has by computing the cosine similarity (Equation (2)) between their vectors. The final relevance score of a new functionality \( f \) to the target \( u \) is given by the weighted average:

\[
\text{Relevance}(f,u) = \frac{\sum_{f' \in F(u)} \text{CosSim}(V_f, V_{f'}) \times C_{u,f'}}{\sum_{f' \in F(u)} \text{CosSim}(V_f, V_{f'})},
\]

where \( F(u) \) is the set of functionalities that the target user \( u \) already has, \( f' \) denotes each functionality inside this set, and \( C_{u,f'} \) is the rating count indicating how many times user \( u \) has rated the functionality \( f' \). For each target user, we select top \( K \) new functionalities having highest relevance scores for him/her.

**Candidate Set Generation.** Given the target user and a list of new functionalities predicted for him/her, this step is to find out those apps providing the new functionalities. We retrieve apps that contain one or more predicted functionalities and have not been rated by the target user, and put these apps into the candidate set. Each app is indexed by the functionalities it provides.

**Candidate Set Filtering.** The candidate set generated by previous step may contains many apps providing overlap functionalities. This step is to filter out those redundant apps by ranking similar apps and selecting the one that most likely satisfies the user’s requirements. We conduct SVD to user ratings that may capture users’ non-functional requirements. SVD maps both users and apps into a latent factor space having \( d \) dimensions. Each latent dimension is associated with one uninterpretable aspect of users’ non-functional requirements (we do not need to know what each aspect exactly is, they can be UI design, ease of use, etc.). Accordingly, each app is represented as a vector \( \tilde{v}_a \in R^d \), in which the values of elements measure the extent to which app \( a \) possesses those aspects. Similarly, each user is represented as a vector \( \tilde{p}_u \in R^d \) in which the values of elements measure the importance of those aspects to user \( u \). The predicted rating of app \( a \) for user \( u \) is given by:

\[
\hat{r}_{u,a} = \mu + b_u + b_a + \tilde{q}_a \cdot \tilde{v}_a + \lambda_a F_{u,a}.
\]

\( \mu \) denotes the overall average rating. \( b_u \) and \( b_a \) indicate the observed deviations of user \( u \) and app \( a \) respectively from the average, with the consideration that some users may tend to rate higher/lower than other users, and some apps may tend to be rated higher/ lower than other apps. In order to give more weights to those apps providing multiple functionalities, we include \( F_{u,a} \) (weighted by \( \lambda_a \)) that indicates the number of functionalities needed by user \( u \) and provided by app \( a \). The model parameters \( (b_u, b_a, \tilde{q}_a, \tilde{p}_u, \lambda_u) \) can be learnt through Gradient Descent Optimization. The details of the optimization algorithm can be found in (Funk 2006).

For each set of apps providing similar functionalities, we use Equation (5) to estimate the rating for each similar app, and select the one having highest estimated rating as recommendation, therefore filtering out similar apps providing overlap functionalities and addressing the redundancy problem.

Since our proposed method employs user reviews and user ratings as input data, data sparsity problem that refers to insufficient input data may exist. Although data sparsity problem has been commonly recognized as one of the hardest problems that cannot be perfectly solved, our method could be efficient to some extent in alleviating the sparsity problem. First, at functionality extraction stage, to alleviate the impact of the newly released apps whose user reviews are very sparse, we use their description instead of using their user reviews to avoid extracting inaccurate functionalities. Second, at the functionality prediction stage, we address the sparsity problem by incorporating the functionalities to enrich the app information and decomposing apps into functionalities, which may reduce the dimensionality of sparse user ratings. Third,
at the app selection stage, we employ SVD that has been widely proved to be an efficient dimensionality reduction technique in alleviating rating sparsity. Moreover, since the functionalities of apps have been explicitly captured in previous stages, and the ratings are only used to capture the non-functional aspects, the impact of rating sparsity is further lowered.

**Preliminary Results**

In this section we will report the results of our preliminary evaluation on the functionality extraction which is a crucial part of our method. The sample data (app descriptions and user reviews) we use is crawled from Apple App Store (U.S.)\(^3\). We construct the vocabulary based on the textual descriptions of 4375 popular apps evenly distributed in 24 categories. The constructed vocabulary contains 9068 words and phrases. The number of similar apps for moderation is 3 (i.e. \(N=3\)), and the moderation parameter \(\alpha\) is set to 0.6. For the sake of space, we only show the results for 5 apps, and for each app, we only list the first 12 extracted functionalities. The preliminary results are shown in Table 1.

<table>
<thead>
<tr>
<th>App Name</th>
<th>Functionalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dropbox</td>
<td>doc, file, space, photo, video, computer iphone, access file, share link, access photo, video device, share photo, attachment</td>
</tr>
<tr>
<td>WhatsApp</td>
<td>message, massager, chat, group, contact, friend, address book, notification, friend world, family, group chat, voice note</td>
</tr>
<tr>
<td>Gmail</td>
<td>gmail, mail, google, conversation, inbox, account support, attachment, get notification, account, mail app, contact, send email</td>
</tr>
<tr>
<td>YouTube</td>
<td>video, playlist, youtube, video playlist, list search, watch video, watch list, share video, channel, search video, share friend, entertainment</td>
</tr>
</tbody>
</table>

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 recommendation architecture, since the functionalities are the basis of further analysis for recommendation. The preliminary results show that our proposed within-app and cross-app analysis is effective in extracting app functionalities with good quality from user reviews, which guarantees the effectiveness of the whole architecture.

**Expected Contribution and Future Work**

In this paper, we propose a functionality-based mobile app recommendation architecture. Our method recommends apps by revealing the detail functionalities of apps and truly capturing users’ functional requirements, which have not been considered by existing works. Besides, we prove that user reviews can be used to enrich item information and can be incorporated to enhance recommendation. Our work has not only theoretical contributions to recommendation literature, but also practical implications. The proposed architecture can be implemented as an effective real-world app recommender system helping users discover apps that meet their requirements. The recommended apps would be more accurate, more diverse, and less overlap in functionalities.

We will complete this work by rigorously evaluating our method on different recommendation evaluation metrics, including Mean Absolute Error (MAE) (Herlocker et al. 2004) for rating prediction accuracy, F-measure and NDCG@k (Järvelin and Kekäläinen 2002) for ranking accuracy, and Intra-List Similarity (Ziegler et al. 2005) for diversity. We will comprehensively compare our method with state-of-the-art methods that have been proved to have best results to demonstrate the superiority of our method. Our experiments will be conducted on three different datasets crawled from three main app stores, i.e. Apple,

\(^3\) https://itunes.apple.com/us/genre/mobile-software-applications/id36?mt=8
Windows\(^4\) and Android\(^5\), to show the robustness of our proposed method. We will also investigate the impact of data sparsity by testing our method upon data with varying level of sparsity.

Our method focuses more on those apps providing functionalities for users. However, there are also apps that may not be functionality-oriented, e.g. games. In future work, we will investigate the impact of app category on user requirement modeling, and extend our work by coming up with strategies to capture user requirements under different app categories. Moreover, we are also considering incorporating other data sources (e.g. user-generated content in social network) to enrich app information, especially for newly released apps.

### References


---

\(^4\) http://windows.microsoft.com/en-us/windows-8/apps

\(^5\) https://play.google.com/store/apps
