Addressing Cold-Start in App Recommendation: Latent User Models Constructed from Twitter Followers

Jovian Lin[1,2] Kazunari Sugiyama[1] Min-Yen Kan[1,2] Tat-Seng Chua[1,2]
[1]School of Computing, National University of Singapore, Singapore
[2]Interactive and Digital Media Institute, National University of Singapore, Singapore
jovian.lin@gmail.com
{sugiyama,kanmy,chuats}@comp.nus.edu.sg

ABSTRACT

As a tremendous number of mobile applications (apps) are readily available, users have difficulty in identifying apps that are relevant to their interests. Recommender systems that depend on previous user ratings (*i.e.*, collaborative filtering, or CF) can address this problem for apps that have sufficient ratings from past users. But for apps that are newly released, CF does not have any user ratings to base recommendations on, which leads to the *cold-start* problem.

In this paper, we describe a method that accounts for nascent information culled from Twitter to provide relevant recommendation in such cold-start situations. We use Twitter handles to access an app's Twitter account and extract the IDs of their Twitter-followers. We create *pseudo-documents* that contain the IDs of Twitter users interested in an app and then apply latent Dirichlet allocation to generate latent groups. At test time, a target user seeking recommendations is mapped to these latent groups. By using the transitive relationship of latent groups to apps, we estimate the probability of the user liking the app. We show that by incorporating information from Twitter, our approach overcomes the difficulty of cold-start app recommendation and significantly outperforms other state-of-the-art recommendation techniques by up to 33%.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information filtering, Search process

General Terms

Algorithms, Experimentation, Performance

Keywords

Recommender systems, Collaborative filtering, Cold-start problem, Mobile apps, Latent user models, Twitter

SIGIR'13, July 28–August 1, 2013, Dublin, Ireland.
Copyright 2013 ACM 978-1-4503-2034-4/13/07 ...$15.00.

1. INTRODUCTION

Mobile applications (apps) for smartphones are soaring in popularity and creating economic opportunities for app developers, companies, and marketers. At present, users are able to access a substantial number of apps via *App Stores*; furthermore, the selection available in app stores is growing rapidly as new apps are approved and released daily (Apple recently reported that more than 40 billion apps have been downloaded for its devices[1]). While this growth has provided users with a myriad of unique and useful apps, the sheer number of choices also makes it more difficult for users to find apps that are relevant to their interests.

Historically, recommender systems have been introduced to alleviate this type of information overload by helping users find relevant items (*i.e.*, apps). One effective and widely used recommender system technique, collaborative filtering (CF), employs user ratings to recommend items. However, CF is ineffective in the case of new apps as they have no prior ratings to base recommendations on. This is an important manifestation of the *cold-start* problem, in which the performance of CF systems suffers for items with few or no prior ratings. The only way for the systems to provide automatic recommendation is to either wait for sufficient ratings to be supplied by users—which will take some time—or use content-based filtering (CBF).

CBF algorithms seek to recommend items based on similar content. For apps, this would be the metadata such as textual descriptions and genre information. An obvious drawback of CBF is that the recommended items are similar to the user's previously-consumed items [23]. For example, if a user has only downloaded weather-related apps, CBF would recommend other apps that are related to weather. This lack of diversity results in unsatisfactory recommendations.

With the rise of social networking services (SNS), people broadcast and message friends, colleagues, and the general public about many different matters in a short and informal manner [25]. They do it often as the overhead of broadcasting such short messages is low. For instance, Twitter experienced several record-breaking events in 2012, such as having one million accounts created daily and having 175 million tweets sent out per day[2]. Additionally, SNS often include rich information about its users [33], such as posted user-generated content, user logs, and user-to-user connections (*e.g.*, Alice *follows* Bob).

People use SNS to talk about many subjects—including apps. An interesting possibility thus arises: Can we merge information

[1] App Store Tops 40 Billion Downloads with Almost Half in 2012. Retrieved Jan 10, 2013, from `http://goo.gl/fvND9`.

[2] 100 Fascinating Social Media Statistics and Figures From 2012. Retrieved Jan 10, 2013, from `http://goo.gl/IvSJW`.

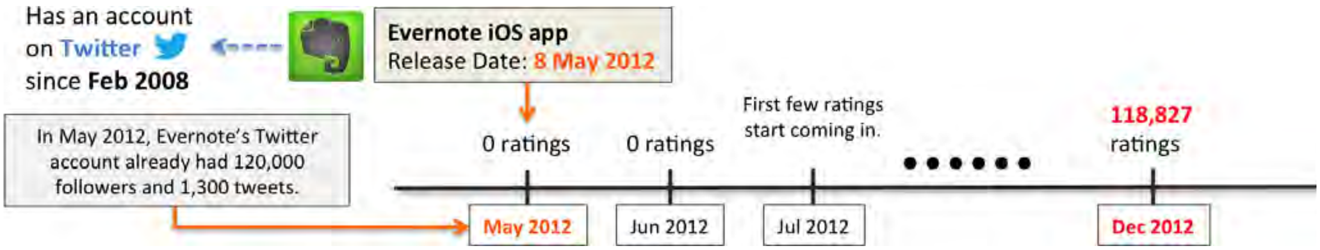


Figure 1: For two months since its release on the iTunes App Store, the Evernote app did not have any ratings. However, its Twitter account already had active tweets and followers. This shows that despite the cold-start, there is still information out there about the app, particularly on social networking services like Twitter.

mined from the rich data in SNS to enhance the performance of app recommendation? More formally, can we address the cold-start weaknesses of the proprietary user models of recommender systems (*e.g.*, the *user profiles* in an app store) by using nascent signals about apps from SNS (*e.g.*, the *user profiles* in Twitter)?

Through our case studies on the correlation and lag between SNS and formal reviews in app stores, we verified that the answer to this question is indeed "yes." How then, would one go about capturing and encoding data from SNS? Through our observation of app stores, we note that the descriptions of some apps contain references to their Twitter accounts; by having a Twitter account, an app's developer or the organization can interact with its users on Twitter and market themselves, such as announcing new apps or updates. For example, within the descriptions in its Google Play[3] and iTunes[4] app stores, the "Angry Birds Star Wars" app has a line that says "follow @angrybirds[5] on Twitter."

We also observed that app mentions in SNS can precede formal user ratings in app stores. This is important as it asserts that an early signal for app ranking (and thus recommendation) can be present in SNS. For example, Figure 1 shows that the "Evernote" app that was released in May 2012 had no ratings in the iTunes App Store for two months. It was only until July 2012 that the first few ratings started coming in. However, even before May 2012, Evernote's Twitter account[6] already had more than 120,000 followers and 1,300 tweets. We take advantage of this active yet indirect information that is present in Twitter and use it to alleviate the cold-start problem that besets newly released apps.

We integrate these findings into a novel approach to app recommendation that leverages on information from SNS (in specific, Twitter) to drive recommender systems in cold-start situations. By extracting *follower* information from an app's (official) Twitter account, we create a set of *pseudo-documents* that contains Twitter-follower information; these pseudo-documents are different from the standard documents written in natural language. We then utilize latent Dirichlet allocation (LDA) [6] to construct latent groups of "Twitter personalities" from the pseudo-documents. By harnessing information from the linked topology of Twitter accounts, we can infer a probability of how likely a target user will like a newly released app—even when it has no official ratings.

We conduct extensive experiments and compare the usage of Twitter-follower information with other types of features, such as app genres, app developers, and text from app descriptions. In order to show that our approach excels not because of the use of Twitter information alone, we compare our approach with a non-LDA technique that also employs the same information from Twitter. Finally, we combine our Twitter-follower feature with other features gleaned from app metadata through the use of gradient boosted trees (GBT) [10] and compare our approach with other state-of-the-art techniques. We show that our approach significantly outperforms these techniques.

2. RELATED WORK

2.1 Addressing the Cold-Start

As the lack of ratings (*i.e.*, the cold-start) hinders the use of CF techniques [29], various alternatives have been employed to overcome the problem. For example, Zhou *et al.* [35] experimented with eliciting new user preferences by using decision trees with collaborative filtering. Moshfeghi *et al.* [21] proposed a method for combining content features such as semantic and emotion information with ratings information for the recommendation task. Liu *et al.* [19] identified representative users whose linear combinations of tastes are able to approximate other users. Likewise, many other works attempt to overcome the cold-start by finding radical ways of using proprietary user models and content.

2.2 Latent Factor Models

Another effective approach to the cold-start is to use latent factor models [17] that map users and items into a dense and reduced latent space that captures their most salient features. These models provide better recommendations than traditional neighborhood methods [11] as they reduce the level of sparsity and improve scalability [16]. Notable latent factor models include probabilistic latent semantic analysis (PLSA) [12], principal component analysis (PCA) [15], neural networks such as the restricted Boltzmann machine (RBM) [27], and singular vector decomposition (SVD) [30]. However, latent factor models have the following two major disadvantages in recommendation tasks: Firstly, the learned latent space is not easy to interpret. Secondly, many latent factor models rely on other user ratings, which may be lacking if the dataset is sparse.

2.3 Probabilistic Topic Models

In order to better understand learned latent groups, a few recent recommender systems employ probabilistic topic models. These topic models [5] discover a set of "topics" from a large collection of documents, where a topic is a distribution over terms that is biased around those associated under a single theme [31]. One of the widely used probabilistic topic models is latent Dirichlet allocation (LDA) [6], which can be seen as a dimensionality reduction technique like latent factor models, but with proper underlying gen-

[3] "Angry Birds Star Wars" in Google Play Store. Retrieved Jan 10, 2013, from http://goo.gl/Dn0T1.

[4] "Angry Birds Star Wars" in iTunes App Store. Retrieved Jan 10, 2013, from http://goo.gl/9ov2K.

[5] http://twitter.com/angrybirds

[6] http://twitter.com/evernote

erative probabilistic semantics that make sense for the type of data that it models.

Topic models provide an interpretable, low-dimensional representation of the documents [7] and may aid in the recommendation process. In recent works, Ramage *et al.* [24] used a variant of LDA to characterize microblogs in order to recommend which users to follow. They used both tweets and hashtags to help identify prominent topics for Twitter users, and used these topics to characterize the users. Wang and Blei [31] combined CF based on latent factor models and content based on probabilistic topic modeling to recommend scientific papers. LDA is also employed in [21], which we described in Section 2.1.

2.4 User Profiling Using Social Networks

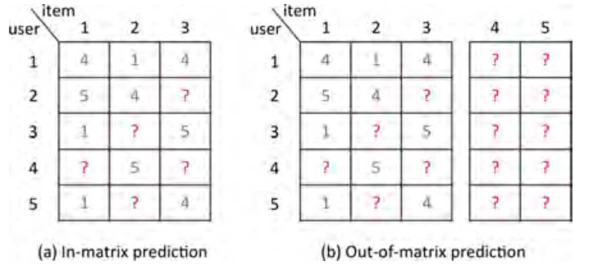
With the emergence of social networks, recommender systems that rely on SNS have started to gain popularity. For example, Said *et al.* [26] explored movie recommendations based on user ratings and a social network (that comes with the user ratings data) where users can befriend one another. Jamali and Ester [14] incorporated trust propagation mechanisms into the matrix factorization technique in order to recommend items. We note that the aforementioned works rely on social networks within the dataset itself; that is, they do not rely on external SNS. As many domains and datasets do not incorporate a social network of their own, it is more realistic if we can glean useful data from an external and ubiquitous source, such as Twitter.

In addition, given an external social network, a standard way of profiling users is through the identification of *influence* [3], which can be measured in a number of ways. Kwak *et al.* [18] compared three different measures of influence on Twitter (number of followers, PageRank [22], and number of retweets) and found that the ranking of the most influential users differed depending on the measure. Similarly, Cha *et al.* [9] also compared three different measures of influence (number of followers, number of retweets, and number of mentions) and found that all three captured different perspectives of influence. We note that "following" information has generally been a popular indicator for measuring influence.

Our work differs from the ones mentioned in Section 2.1 in that instead of using proprietary user models and content, we use information from Twitter (*i.e.*, non-proprietary content from SNS) and construct latent groups of "Twitter personalities" to predict recommendations under the cold-start. Although textual features have generally been a popular source of alternative data to substitute for the lack of ratings—such that even state-of-the-art techniques (Section 2.2 and 2.3) are primarily dependent on—it is not a universal solution as not all domains contain reliable textual data. Additionally, as mentioned in Section 2.4, it is more realistic to rely on external SNS. Therefore, our work is unique in that it uses the "who *follows* whom" information on Twitter as its primary source of data, as textual features in the app domain are inherently noisy and unreliable (in contrast to the cases of movies or scholarly papers).

3. OUR APPROACH

We first explain the kind of problem we address. Then we describe the relation between apps and Twitter-followers, and how we use them in our work. Next, we construct *pseudo-documents* and *pseudo-words* using data from users, apps, users' preferences, and Twitter-followers. Thereafter, we generate *latent groups* from the pseudo-documents, whereby a latent group represents the combined "interests" of various Twitter-followers. Finally, the set of latent groups is used as a crucial component of our algorithm to estimate the probability of a target user liking an app.



user \ item	1	2	3
1	4	1	4
2	5	4	?
3	1	?	5
4	?	5	?
5	1	?	4

(a) In-matrix prediction

user \ item	1	2	3	4	5
1	4	1	4	?	?
2	5	4	?	?	?
3	1	?	5	?	?
4	?	5	?	?	?
5	1	?	4	?	?

(b) Out-of-matrix prediction

Figure 2: Difference between (a) in-matrix prediction and (b) out-of-matrix prediction.

3.1 Targeting the Cold-Start Problem

There are two types of cold-start problems in CF: in-matrix prediction and out-of-matrix prediction [31]. Figure 2(a) illustrates in-matrix prediction, which refers to the problem of recommending items that have been rated by at least one user in the system. This is the task that CF researchers have often addressed [8, 1, 13, 28, 34, 4, 27, 17].

On the other hand, Figure 2(b) illustrates out-of-matrix prediction, where newly released items (*e.g.*, items 4 and 5) have never been rated. Traditional CF algorithms cannot predict ratings of items in the out-of-matrix prediction as they rely on user ratings, which are unavailable in this scenario.

Our work focuses on this second, more challenging problem. Hereafter, we use "cold-start problem" to refer to "out-of-matrix prediction," for ease of reference.

3.2 Apps and their Twitter-Followers

Apps have textual descriptions displayed in their app store entries; some of these descriptions further contain links to the apps' official Twitter account (*i.e.*, *Twitter handle*). For example, the "Angry Birds" franchise contains a link to its Twitter handle (@angrybirds). From the handle, we can identify the IDs of Twitter-followers who follow the app. We note that by following an app's Twitter handle, the Twitter-followers subscribe to the tweets that are related to the particular app, which can be seen as an indicator of interest [9]. Figure 3 illustrates the relation between users, apps, and Twitter-followers. By using information from the apps' Twitter-followers, we can construct "latent personalities" from two sources of data: the app store and Twitter. Using these latent personalities, our algorithm is able to recommend newly released apps in the absence of ratings[7] as shown in Figure 2(b).

Given that an app has a set of Twitter-followers, our approach estimates the probability that user u—defined by his or her past ratings—likes app a that is *followed* by Twitter-follower t (*i.e.*, Twitter-follower t follows app a's Twitter account):

$$p(+|t, u), \tag{1}$$

where "$+$" denotes the binary event that a user likes an app.

Furthermore, as an app is represented by a set of its Twitter-followers, it is necessary to aggregate the probability in Equation (1) over the various Twitter-followers of app a. This allows us to estimate the probability of how likely the target user will like the app:

$$p(+|a, u). \tag{2}$$

[7] Note that although other types of information can be extracted from Twitter, such as the textual tweets and hashtags, we only focus on the Twitter-followers in this work as early experiments have shown that other types of information are noisy and potentially ambiguous.

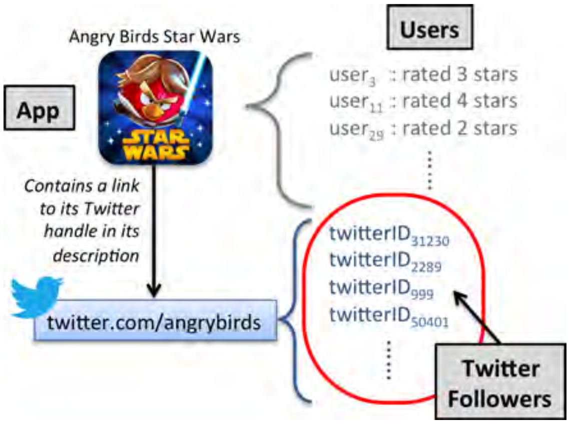


Figure 3: Instead of solely relying on the ratings of users, our approach also makes use of the Twitter IDs that *follow* the apps (red oval).

3.3 Pseudo-Documents and Pseudo-Words

In order to estimate the probability $p(+|a, u)$ in Equation (2), we build upon latent Dirichlet allocation (LDA)—a generative probabilistic model for discovering latent semantics that has been mainly used on textual corpora. Given a set of textual documents, LDA generates a probability distribution over latent "topics" for each document in the corpus, where each topic is a distribution over words. Documents that have similar topics share the same latent topic distribution.

We adapt LDA for the purpose of collaborative filtering. As mentioned in Section 3.2, users download apps and apps may have Twitter-followers. Hence, user u and the Twitter-followers (of the apps that user u has downloaded) are analogous to a document and the words in the document, respectively. For the sake of clarity, we will call them *pseudo-documents* from now on as our "documents" actually do not contain natural language words in our work.

Drawing on the parallelism, we formally define the following terms:

- We first assume that user u likes app a, and app a has a set of Twitter-followers $\{t_1, \ldots, t_n\}$ following its Twitter handle.
- A *pseudo-document* represents user u. Because of this, we use u to represent both pseudo-document and user.
- A *pseudo-word* is a "word" in pseudo-document u that corresponds to the ID of a Twitter-follower t. If user u has liked the apps a_1, a_2, and a_3, the pseudo-document u will then contain all the IDs of the Twitter-followers that are following the Twitter handles of apps a_1, a_2, and a_3. Note that there may be duplicated *pseudo-words* as a Twitter-follower t may be following more than one app's Twitter handle.

However, the problem of only considering the "liked" apps is in that LDA will indirectly assign higher probabilities to apps that many users liked. In other words, LDA will indirectly give high probabilities to popular apps. For example, suppose that two apps are judged the same number of times. The probability given by LDA will rank the two apps in order of their probability to be "liked," which we desire. In contrast, if the first app has been judged by all the users and half of them liked the app, it will have the same probability as another app that was judged by only half of the users but liked all the time, which is undesirable.

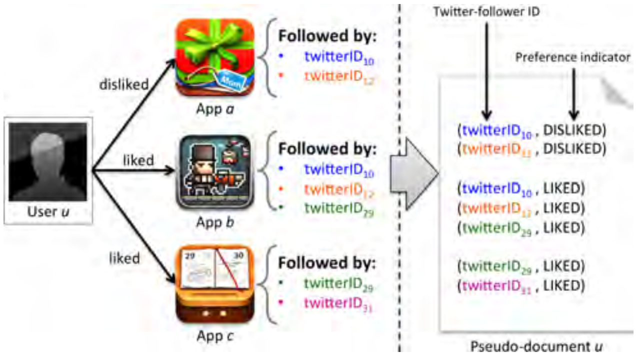


Figure 4: A *pseudo-document* is constructed based on information from a user, apps, Twitter-followers, and binary ("liked" or "disliked") preference indicators. A pseudo-document contains *pseudo-words*; each pseudo-word is represented as a *tuple* containing a Twitter-follower ID and a binary preference indicator.

To address this popularity problem, we incorporate the magnitude of negative information as it indirectly allows us to account for the frequency of the whole judging group (*i.e.*, "liked" apps + "disliked" apps = total apps). In addition, this solution allows user groups to not only reflect the Twitter-followers that appear in the apps that they like, but also in the apps that they dislike, thus providing richer information. We now formally define a pseudo-word:

- A *pseudo-word* is a "word" in pseudo-document u that contains the ID of a Twitter-follower t and its associated binary ("liked" or "disliked") preference indicator.

Figure 4 illustrates how a *pseudo-document* is constructed based on *pseudo-words*, which in turn are constructed based on the IDs of Twitter-followers and the binary preference indicators from users. Furthermore, in order to obtain the binary preferences, it is mandatory to convert the user ratings (*i.e.*, the 5-point Likert scale) into binary like/dislike indicators (see Section 4.2).

Note that the concept of pseudo-documents and pseudo-words does not apply exclusively to Twitter-followers; it can also be applied to other sources of information such as the app genres, app developers, and the words in the app descriptions. For example, instead of using the IDs of Twitter-followers, we can also construct pseudo-words based on the IDs of app developers. We focus on Twitter-followers as our goal is to assess the effectiveness of this new source of data. In Section 5.1, we will create different sets of pseudo-documents based on different features (*i.e.*, Twitter-followers, app genres, app developers, and words) and identify the most effective feature in the recommendation of apps.

3.4 Constructing Latent Groups

Given the set of pseudo-documents $\{u_1, \ldots, u_m\}$, LDA can generate a probability distribution over *latent groups* for each pseudo-document, where each latent group is represented as a distribution over Twitter-follower IDs and binary preference indicators. Figure 5 illustrates this framework.

By using the information on "which apps are followed by which Twitter-followers," we can estimate the probability of target user u liking app a by taking into account the IDs of the Twitter-followers following app a, and marginalizing over the different latent groups of pseudo-document u.

Formally speaking, given the set of tuples of pseudo-words $T_u = \{(t_1, d_1), \ldots, (t_n, d_n)\}$, where t_i and d_i are a Twitter-follower ID

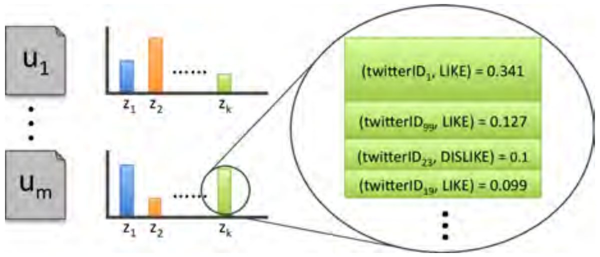

u1
um
z1
z2
zk
(twitterID1, LIKE) = 0.341
(twitterID99, LIKE) = 0.127
(twitterID23, DISLIKE) = 0.1
(twitterID19, LIKE) = 0.099

Figure 5: Given the set of pseudo-documents $\{u_1, \ldots, u_m\}$, LDA generates a probability distribution over *latent groups* for each pseudo-document, where each latent group z is represented as a distribution over pseudo-words. A pseudo-word is represented as a tuple containing a Twitter-follower ID and a binary preference indicator.

and its associated binary ("liked" or "disliked") preference indicator, respectively, we define LDA as a generative process that creates a series of tuples:

$$p(T_u|\alpha, \beta) = \int p(\theta|\alpha) \left(\prod_{i=1}^{n} \sum_{z=1}^{K} p(z|\theta) p(t_i, d_i|\beta_z) \right) \mathrm{d}\theta,$$

where K is the number of latent groups, θ follows the Dirichlet distribution of hyper-parameters α, and the latent group z follows a multinomial distribution given by β_z. The model is fully specified by α and β_z for each possible latent group z. Those hyper-parameters are learned by maximizing the likelihood of the dataset.

The LDA model is used to compute the probability that the presence of a Twitter-follower t indicates that it is "liked" $(+)$ (or "disliked" $(-)$) by user u, given user u's past interaction T_u and the learned parameters α and β. Hence, we get the following equation:

$$\begin{aligned} p(\pm, t|u) &= p(\pm, t|T_u, \alpha, \beta) \\ &= \sum_{z \in Z} p(\pm, t|z) p(z|u), \end{aligned} \tag{3}$$

where Z is the set of latent groups, $p(\pm, t|z)$ is computed from the per-topic word distribution of LDA, and $p(z|u)$ is computed from the per-document topic distribution of LDA.

3.5 Estimation of the Probability of How Likely the Target User Will Like the App

Our approach is based on a simple "averaging" method where the probability of how likely the target user will like the app is the expectation of how the Twitter-followers like the app. Given a set of Twitter-followers T, the probability that user u likes app a is defined as follows:

$$\begin{aligned} p(+|a, u) &= \sum_{t \in T(a)} p(+, t|a, u) \\ &= \sum_{t \in T(a)} p(+|t, u) P(t|a), \end{aligned} \tag{4}$$

where $T(a)$ is the set of possible Twitter-followers following app a, in which we assume that: (i) Twitter-followers are examined one at a time to make a decision about whether an app is liked or disliked. In this case, t and t' are disjoint events whenever $t \neq t'$; (ii) when the Twitter-follower is known, the judgement does not depend on the app any more, *i.e.*, $p(+|t, a, u) = p(+|t, u)$; (iii) the fact that given a user and an app, there is no judgement involved, *i.e.*, $p(u, a) = p(u)p(a)$; (iv) the fact that an app has a given Twitter-follower is independent from the user, *i.e.*, $p(t|a, u) = p(t|a)$.

Equation (4) is then reduced to the estimation of two quantities:

1. the probability that user u likes app a given that app a has Twitter-follower t, *i.e.*, $p(+|t, u)$, and
2. the probability of considering Twitter-follower t given app a, *i.e.*, $p(t|a)$.

$p(+|t, u)$ is straightforward to estimate as it can be rewritten as:

$$p(+|t, u) = \frac{p(+, t|u)}{p(+, t|u) + p(-, t|u)},$$

where $p(+, t|u)$ and $p(-, t|u)$ are derived from LDA in Equation (3), which is the probability that Twitter-follower t occurs in an app that is liked (or disliked) by user u.

As for the probability $p(t|a)$, we explore the following two ways:

(I) Assume a uniform distribution over the various Twitter-followers present in app a

This framework is defined as follows:

$$p(t|a) = \begin{cases} \frac{1}{\#T(a)} & \text{if } t \text{ is a Twitter-follower of app } a \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

where $T(a)$ and $\#T(a)$ denote the set of Twitter-followers following app a and the set cardinality, respectively.

(II) Give more importance to influential Twitter-followers

In this alternative framework, we compute the influence of Twitter-followers by using TwitterRank [32]. Let $TR(t)$ be the TwitterRank score of Twitter-follower t. With that, for each Twitter-follower t, we retrieve its TwitterRank score and normalize it with the scores obtained by the other Twitter-followers following app a. That is:

$$p(t|a) = \begin{cases} \frac{TR(t)}{\sum_{t' \in T(a)} TR(t')} & \text{if } t \text{ is a Twitter-follower of app } a \\ 0 & \text{otherwise,} \end{cases} \tag{6}$$

where $T(a)$ denotes the set of Twitter-followers following app a.

Prior tests on our dataset have shown that there is no significant difference in performance between (I) and (II) above. This is because the Twitter-followers that we get (note that each latent group consists of Twitter-followers) are generally not prominent, influential people. Rather, they are the average users on Twitter. Therefore, even if we use TwitterRank, the influence of each Twitter-follower eventually converges into the uniform distribution. We thus adopt the uniform distribution defined by Equation (5) in our evaluation for its simplicity.

4. EVALUATION PRELIMINARIES

We preface our evaluation proper by detailing how we constructed our dataset, our settings for the dataset and the LDA model, and how we chose our evaluation metric.

4.1 Dataset

We constructed our dataset by crawling from Apple's iTunes App Store and Twitter during September to December 2012. The dataset consists of the following three elements:

1. *App Metadata.* These include an app's ID, title, description, genre, and its developer ID. The metadata is collected by first getting all the app IDs from the iTunes App Store, and then retrieving the metadata for each app via the iTunes Search API. The metadata is the source for the genre, developer, and word features mentioned at the end of Section 3.3.

2. *Ratings*. For each app, we built a separate crawler to retrieve its reviews from the iTunes App Store. A review contains the app's ID, its rating, the reviewer's ID, the subject, and the review comments. This is the source of the rating feature.

3. *Related Twitter IDs*. We used two methods to collect app-related Twitter IDs. The first way is to get the IDs that follow an app's Twitter account. We scanned through each app's description to identify its Twitter handle. For each app's Twitter handle, we used Twitter's API to search for every Twitter ID following its handle. The second method uses Twitter's Streaming API to receive tweets in real-time. To retrieve tweets that are related to apps, we only kept tweets that contain hyperlinks to apps in the iTunes App Store, and stored the Twitter IDs (who wrote the tweet) as well as the app IDs that were mentioned in the tweets.

Altogether, we collected 1,289,668 ratings for 7,116 apps (that have accounts on Twitter) from 10,133 users. The user-item ratings matrix has a sparsity of 98.2%. We also collected 5,681,197 unique Twitter IDs following the apps in our dataset. With respect to app ratings and the number of related Twitter-followers per app, we restrict that each user gives at least 10 ratings and each Twitter ID is related to at least 5 apps, respectively. On average, each user has rated 26 apps—ranging from a minimum of 10 rated apps to a maximum of 271.

4.2 Experimental Settings

In order to train the LDA model, we require binary relevance judgements to convert the user ratings to binary preference indicators, as well as two sets of hyper-parameters—namely, the number of latent groups K and the priors α and β.

We performed per-user normalization of the 5-point Likert scale ratings when converting them to the binary like/dislike values required by our LDA application. This is because the average rating for different users can vary significantly [16].

Let $r_{avg(u)}$ and $r_{u,a}$ be user u's average rating among all apps and user u's rating for app a, respectively. The normalized rating is thus $r_{n(u,a)} = r_{u,a} - r_{avg(u)}$, where if $r_{n(u,a)} \geq 0$, the rating is converted to a "like" (+), or "dislike" (−), otherwise.

We set the priors α and β as proposed in [20]. The number of latent groups has a crucial influence on the performance of the LDA approach. We used the likelihood over a held out set of training data to find the relevant number of latent groups. We tried several settings for K (*i.e.*, the number of latent groups), namely 10, 20, 35, 50, 80, 100, 120, 150, and 200. The final number of latent groups was selected by maximizing the likelihood of observations over the development set.

In order to simulate the cold-start, we selected 10% of the apps which were then the held out set for all users (*i.e.*, we removed their ratings in the training set). Therefore, each user has the same set of within-fold apps and we can guarantee that none of these apps are in the training set of any user.

We performed 10-fold cross validation where in each fold, we used 70% of the apps to train the LDA model, 20% to identify the number of latent groups for LDA (*i.e.*, the development set), and the remaining 10% as test data.

4.3 Evaluation Metric

Our system outputs M apps for each test user, which are sorted by their probability of liking. We evaluate the algorithms based on which of these apps were actually downloaded and liked (*i.e.*, the normalized rating of the test user).

This methodology leads to two possible evaluation metrics: precision and recall. However, a missing rating in training is ambiguous as it may either mean that the user is not interested in the app (negative), or that the user does not know about the app (truly missing). This makes it difficult to accurately compute precision [31]. But since the known ratings are true positives, we feel that recall is a more pertinent measure as it only considers the positively rated apps within the top M, *i.e.*, a high recall with a lower M will be a better system. We thus chose Recall@M as our primary evaluation metric. Let n_u and N_u represent the number of apps the user likes in the top M and the total number of apps the user likes, respectively. Recall@M is then defined as their ratio: n_u/N_u. We compare systems using average recall, where the average is computed over all test users.

5. EXPERIMENTS

As our approach specifically attempts to address the cold-start, we work on the specific scenario when a new set of apps is released and users have yet to rate them (shown in Figure 2(b)). The goal of our experiments is to answer the following research questions:

RQ1 How does the performance of the Twitter-followers feature compare with other features, such as the app genres, app developers, and words in the app descriptions? Can we obtain better recommendation accuracy by combining them?

RQ2 How does our proposed method compare with other state-of-the-art techniques?

RQ3 Do the latent groups make any sense? If so, what can we learn from them?

5.1 Comparison of Features (RQ1)

We benchmark how the signal from Twitter-followers affects performance in comparison to other sources of data. We compare the Twitter-followers feature (hereafter, T) with the other features: app genres (G), app developers (D), and words in the app descriptions (W). To perform this comparison in a fair manner, for each of these four features, we constructed a set of pseudo-documents that contains a set of *feature-related* pseudo-words (see Section 3.3).

Additionally, we assessed the effectiveness of these features in combination; we combine multiple features (*i.e.*, Twitter-followers, genres, developers, and words) through the use of gradient boosted trees (GBT) [10]. We also performed ablation testing where we removed features from the combined feature set to determine the importance of each feature. The features given to GBT are a set of probabilities defined by Equation (4). In GBT, we set the maximum number of trees and maximum tree depth to 2000 and 3, respectively, and used the least-squares regression as a cost function.

Results. Figure 6 shows the results of the first experiment, which compares the overall performance between features (words (W), developers (D), genres (G), Twitter-followers (T), and all features (All) combined) when we vary the number of returned apps $M = 20, \ldots, 200$. Fixing $M = 100$, we ablated individual features from our combined method and show the results in Table 1.

The results are quite consistent. In the overall comparison over all ranges of M, the individual feature of Twitter-followers gives the best individual performance, followed by genres, developers, and words in the app descriptions. From the combination and ablation study in Table 1, we see that all features are necessary for the optimal results. Matching the results in Figure 6, Table 1 also shows that the removal of the best (worst) individual features leads to the corresponding largest (smallest) drop in recall.

It may be surprising that the developer (D) and genre (G) features are more effective than words in the app description (W).

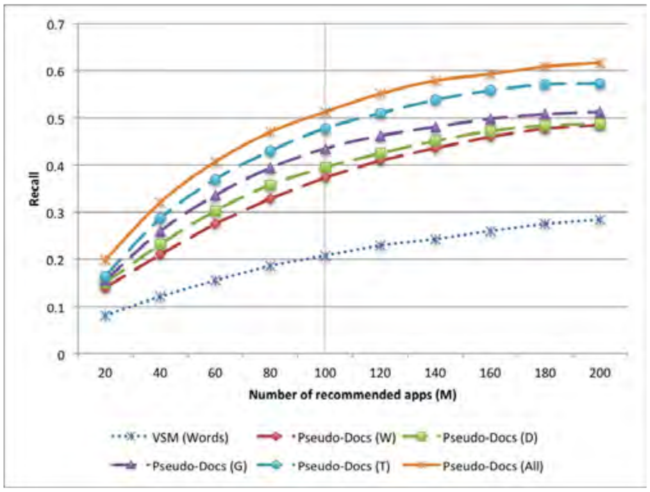


Figure 6: Recall obtained by different individual features (dashed lines), as well as our method that combines all features (solid). The baseline vector space model (VSM), using the app description word vocabulary is also shown (dotted). The vertical line marks model performance at $M = 100$ (*cf.* Table 1).

Table 1: Recall levels in our feature ablation study at $M = 100$. TGDW and individual feature (T, G, D, W) performances in Figure 6 are also shown.

Feature	R@100
All features (TGDW)	**0.513**
All, excluding Twitter-followers (GDW)	0.452
All, excluding Genres (TDW)	0.491
All, excluding Developers (TGW)	0.498
All, excluding Words (TGD)	0.507
Twitter-followers (T)	0.478
Genres (G)	0.435
Developers (D)	0.395
Words (W)	0.373

We observe that users may favor developer brands; possible causes could be that the user recognizes the brand, or that the apps themselves may promote sister apps that are made by the same developer. In addition, some apps complement one another. For example, "Google Chrome," "Gmail," and "Google Maps" form a complementary set. Also, the genres of apps may correlate with download behavior. Figure 7 shows the distribution of app genres, and indeed we observe that the "Games" genre dominates the distribution. This indicates that users often download apps that belong to the "Games" genre.

From a practical standpoint, the most straightforward way to recommend apps in a cold-start would be to use the textual descriptions in a content-based filtering system, as in our (W) system. But it performs the worst among the four individual features. Why do words in app descriptions (W) perform the least well?

Carrying out a more detailed inspection, we find that app descriptions do not give informative hints about the app's role; rather, they are more focused on self-promotion as many apps boast about the reviews that they have received in their app descriptions. Figure 8 is a screenshot of an app description that demonstrates this fact. In addition, according to [2], users pay more attention to screenshots instead of descriptions, which suggests that the information from app descriptions is not as useful. This result also further explains why text-dependent baselines such as CTR and LDA did not perform much better (*cf.* the next section on RQ2).

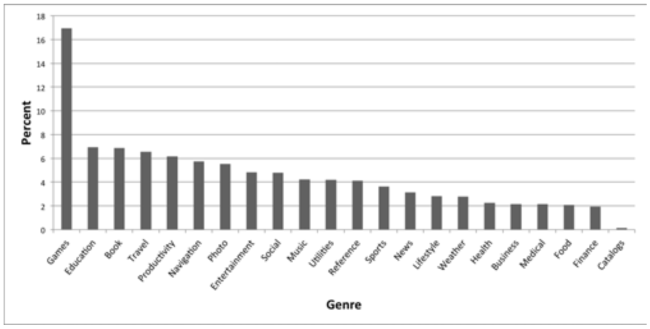
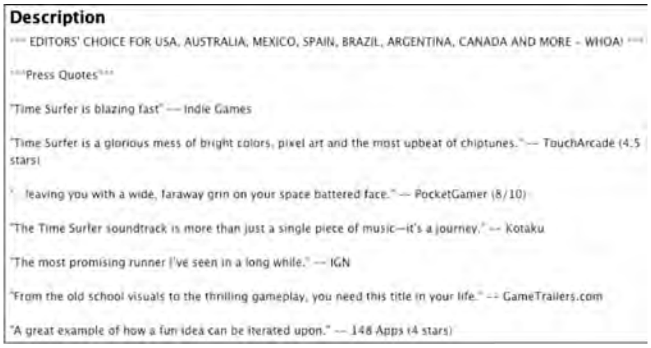


Figure 7: Distribution of app genres within our dataset.


Description

*** EDITORS' CHOICE FOR USA, AUSTRALIA, MEXICO, SPAIN, BRAZIL, ARGENTINA, CANADA AND MORE - WHOA! ***

Press Quotes

"Time Surfer is blazing fast" — Indie Games

"Time Surfer is a glorious mess of bright colors, pixel art and the most upbeat of chiptunes." — TouchArcade (4.5 stars)

"... leaving you with a wide, faraway grin on your space battered face." — PocketGamer (8/10)

"The Time Surfer soundtrack is more than just a single piece of music—it's a journey." — Kotaku

"The most promising runner I've seen in a long while." — IGN

"From the old school visuals to the thrilling gameplay, you need this title in your life." -- GameTrailers.com

"A great example of how a fun idea can be iterated upon." — 148 Apps (4 stars)

Figure 8: A screenshot of an app description that illustrates why word features may not be effective as it largely boasts about endorsements received.

Features aside, we also assessed whether our use of the LDA-based pseudo-document method is an important factor in recall performance. We compared a straightforward use of the same data with a standard vector space model (VSM) where we used the same app description words (W) to build a vector of standard $tf.idf$ weighted words to represent each app.

Figure 6 also shows this result in the bottom two lines. We see that the pseudo-document use of words (W) greatly outperforms the VSM-based version (VSM). A two-tailed t-test at $M = 100$ shows that the improvement is statistically significant ($p < 0.01$). This validates our LDA-based pseudo-document approach.

5.2 Comparison Against Baselines (RQ2)

As we focus on the cold-start problem, we did not consider other well-known recommender techniques that require user ratings, such as matrix factorization or latent factor models. We compare our approach with four baselines. The first two baselines are VSM-based—the first is the app description-based VSM recapped from RQ1 (*i.e.*, "VSM (Words)") while the second uses the IDs of Twitter-followers as its vocabulary (*i.e.*, "VSM (Twitter)"). The VSM (Twitter) baseline evaluates whether our LDA-based approach of using the Twitter-follower data betters the simpler VSM method. The third baseline is the LDA model, which is equivalent to using our pseudo-documents model on words in app descriptions. Lastly, as a much stronger baseline, we show the performance of a re-implementation of Wang and Blei's [31] collaborative topic regression (CTR) model—a state-of-the-art CF algorithm that can also make predictions in cold-start scenarios.

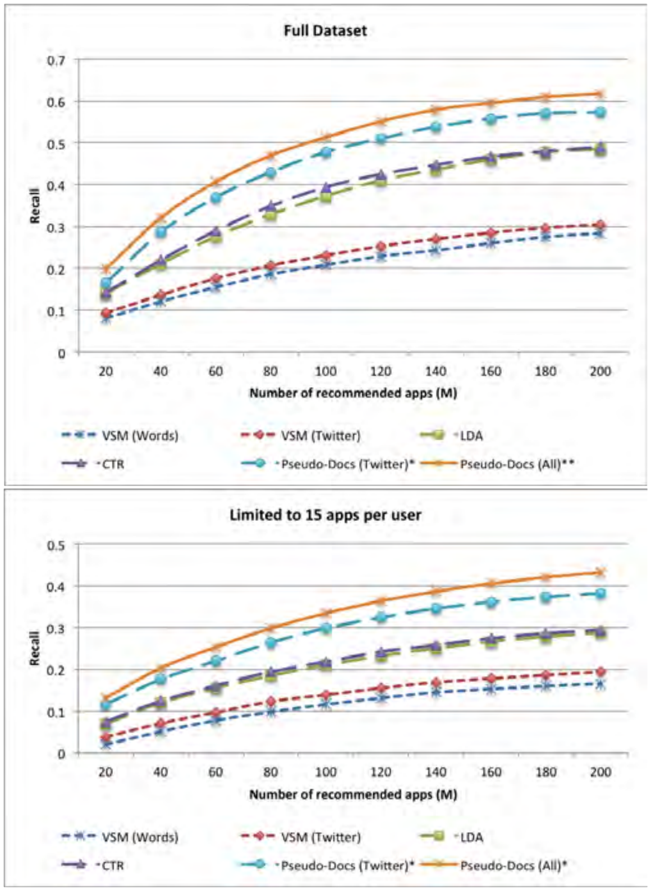


Figure 9: Recall varying the number of recommendations on the full (top) and sparse (bottom) datasets. "*" and "" denote statistically significant improvements over the best baseline (CTR) at $p < 0.05$ and $p < 0.01$, respectively.**

Additionally, in order to study the impact of sparsity on our models, in a separate experimental condition, we randomly removed some ratings from the training set so that the maximum number of rated apps per user was 15, which represents a sparser environment.

Results. Figure 9 plots the results of this set of experiments when we vary the number of returned apps $M = 20, \ldots, 200$. We observe that our approach that combines various pseudo-documents using GBT consistently and significantly outperforms other models (at $p < 0.01$ on the full dataset), particularly the CTR model that is the best among all baselines. This shows that we can achieve significant improvement in recommendation accuracy by integrating multiple sources of information. We also observe that our model of using pseudo-documents with Twitter data alone (*i.e.*, "Pseudo-Docs (Twitter)" in Figure 9) outperforms other models. This validates our observation that Twitter is indeed a good source of information to address the cold-start in app recommendation. It also indirectly points out that the textual features are less effective in the app domain, which is obvious from the performances of the CTR and LDA models that solely rely on textual data. We also note that the performance of CTR is fairly similar to LDA. This is due to the fact that the matrix-factorization component of CTR cannot perform recommendation in the cold-start. Therefore, like LDA, its recommendation is based entirely on content.

We also observe that between the two models that only use Twitter features (*i.e.*, "VSM (Twitter)" and "Pseudo-Docs (Twitter)"), our model significantly outperforms the VSM model. This again validates our earlier findings that our method's performance is not just due to the use of new data, but also how we make use of it.

Another important observation is that under a sparser cold-start environment where we limit our training data to 15 apps per user at maximum, there is an overall drop in recall, which is expected. However, we note that our model (both using Twitter feature alone and multi-features) still outperforms all of the baselines. This indicates that using Twitter features is more robust than using app descriptions under sparse conditions, as the IDs of the Twitter-followers are independent from the apps—misleading or absent app descriptions do not directly influence the Twitter-followers.

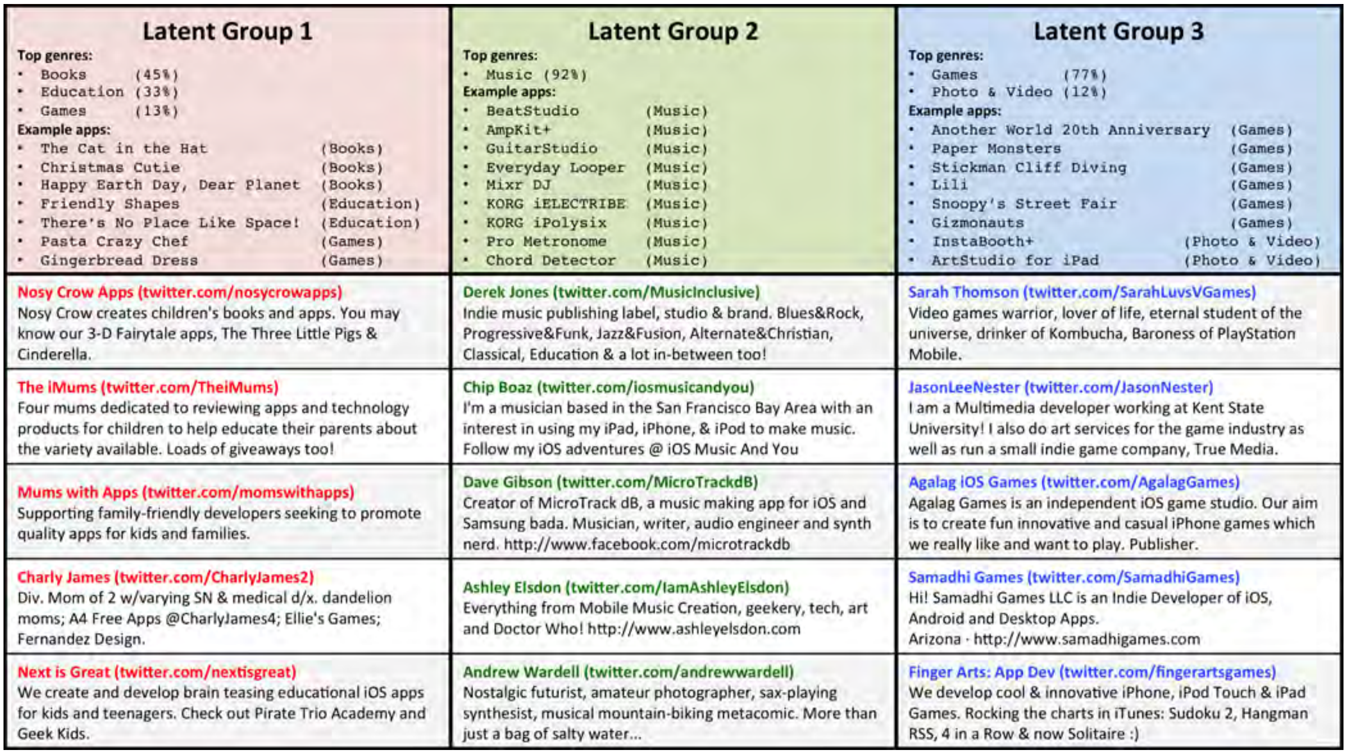
5.3 Analysis of Latent Groups (RQ3)

We use LDA and the notion of Twitter users as pseudo-words to achieve good recall. A natural question to ask is whether the latent groups discovered by LDA (from the individual features of Twitter-followers, genres, and developers) have any meaning. We observe interesting points for the developer (D) and Twitter-follower (T) features.

For the developer feature, in most latent groups, the developers who have received a substantial number of ratings for all their apps are different from the independent ("indie") developers. This is due to the fact that these developers usually have apps that users "liked," whereas the indie developers sometimes have apps that users "disliked" as well as apps that users "liked." This coincides with our hypothesis about brand loyalty, as the developers with a substantial number of ratings either have created a large number of apps (such as EA Games) or have created a small number of well-received apps (such as the apps by Google). We also observe that the clustered developers are classified into the same genre or strong competitors like Facebook and Twitter.

In order to understand why Twitter-followers work best, we scrutinized the latent groups discovered by LDA at the optimal performance point of $K = 120$. Each group consists of Twitter-followers; and each Twitter-follower follows at least 5 apps in our dataset. We then manually visited the Twitter pages of the top 5 Twitter-followers in each of the 120 latent groups, and then verified their profile descriptions and their latest tweets. We observe that more than 95% of these Twitter-followers exhibit consistent interest in apps. This shows that our approach accurately distinguishes between Twitter users who have implicit/explicit interest in apps and regular Twitter users. Our approach assigns low probabilities to Twitter users who have little or no correlation to apps, effectively filtering out their influence as noise.

To illustrate the quality of the latent groups, we performed an additional level of micro-analysis. We selected the 3 most important latent groups among the 120 groups based on the expectation of the probability of each latent group over the set of training data (*i.e.*, the pseudo-documents), along with their corresponding top 5 Twitter individuals' public profile (Figure 10). The top-most row in Figure 10 shows the top genres and examples of apps in each of the top 3 latent groups. We see that the apps in each group coincides with the Twitter profiles of the top 5 individuals in the same group. Latent Group 1 is composed of family-oriented Twitter users who also download family-oriented apps. Such a latent group would be difficult to describe if we were to use genres alone, as this group consists of a non-discrete mix of children-friendly apps. In Latent Group 2, we see that this group consists of professional music-creation apps, which also coincides with the type of Twitter users who are either actual musicians or people interested



Latent Group 1	Latent Group 2	Latent Group 3
Top genres: • Books (45%) • Education (33%) • Games (13%) **Example apps:** • The Cat in the Hat (Books) • Christmas Cutie (Books) • Happy Earth Day, Dear Planet (Books) • Friendly Shapes (Education) • There's No Place Like Space! (Education) • Pasta Crazy Chef (Games) • Gingerbread Dress (Games)	**Top genres:** • Music (92%) **Example apps:** • BeatStudio (Music) • AmpKit+ (Music) • GuitarStudio (Music) • Everyday Looper (Music) • Mixr DJ (Music) • KORG iELECTRIBE (Music) • KORG iPolysix (Music) • Pro Metronome (Music) • Chord Detector (Music)	**Top genres:** • Games (77%) • Photo & Video (12%) **Example apps:** • Another World 20th Anniversary (Games) • Paper Monsters (Games) • Stickman Cliff Diving (Games) • Lili (Games) • Snoopy's Street Fair (Games) • Gizmonauts (Games) • InstaBooth+ (Photo & Video) • ArtStudio for iPad (Photo & Video)
Nosy Crow Apps (twitter.com/nosycrowapps) Nosy Crow creates children's books and apps. You may know our 3-D Fairytale apps, The Three Little Pigs & Cinderella.	Derek Jones (twitter.com/MusicInclusive) Indie music publishing label, studio & brand. Blues&Rock, Progressive&Funk, Jazz&Fusion, Alternate&Christian, Classical, Education & a lot in-between too!	Sarah Thomson (twitter.com/SarahLuvsVGames) Video games warrior, lover of life, eternal student of the universe, drinker of Kombucha, Baroness of PlayStation Mobile.
The iMums (twitter.com/TheiMums) Four mums dedicated to reviewing apps and technology products for children to help educate their parents about the variety available. Loads of giveaways too!	Chip Boaz (twitter.com/iosmusicandyou) I'm a musician based in the San Francisco Bay Area with an interest in using my iPad, iPhone, & iPod to make music. Follow my iOS adventures @ iOS Music And You	JasonLeeNester (twitter.com/JasonNester) I am a Multimedia developer working at Kent State University! I also do art services for the game industry as well as run a small indie game company, True Media.
Mums with Apps (twitter.com/momswithapps) Supporting family-friendly developers seeking to promote quality apps for kids and families.	Dave Gibson (twitter.com/MicroTrackdB) Creator of MicroTrack dB, a music making app for iOS and Samsung bada. Musician, writer, audio engineer and synth nerd. http://www.facebook.com/microtrackdb	Agalag iOS Games (twitter.com/AgalagGames) Agalag Games is an independent iOS game studio. Our aim is to create fun innovative and casual iPhone games which we really like and want to play. Publisher.
Charly James (twitter.com/CharlyJames2) Div. Mom of 2 w/varying SN & medical d/x. dandelion moms; A4 Free Apps @CharlyJames4; Ellie's Games; Fernandez Design.	Ashley Elsdon (twitter.com/IamAshleyElsdon) Everything from Mobile Music Creation, geekery, tech, art and Doctor Who! http://www.ashleyelsdon.com	Samadhi Games (twitter.com/SamadhiGames) Hi! Samadhi Games LLC is an Indie Developer of iOS, Android and Desktop Apps. Arizona · http://www.samadhigames.com
Next is Great (twitter.com/nextisgreat) We create and develop brain teasing educational iOS apps for kids and teenagers. Check out Pirate Trio Academy and Geek Kids.	Andrew Wardell (twitter.com/andrewwardell) Nostalgic futurist, amateur photographer, sax-playing synthesist, musical mountain-biking metacomic. More than just a bag of salty water...	Finger Arts: App Dev (twitter.com/fingerartsgames) We develop cool & innovative iPhone, iPod Touch & iPad Games. Rocking the charts in iTunes: Sudoku 2, Hangman RSS, 4 in a Row & now Solitaire :)

Figure 10: The top 3 latent groups; each group shows the top 5 Twitter-followers and their public profile.

in creating music. This is in contrast with the "Music" genre in app stores, which is in some sense too diverse as it may refer to music-player apps, music-streaming apps, or trivial music-making apps. Lastly, Latent Group 3 captures games that are of a light-hearted, indie nature. We can also relate the Twitter users to the downloaded apps, which are difficult to describe using genres or words alone. In short, our Twitter-follower feature is able to capture the personalities of Twitter-individuals. Therefore, even when an app does not have user ratings, our system can still provide relevant recommendation based on information about *who is following* the app's Twitter account.

6. CONCLUSION

By taking advantage of the unique property of apps and their corresponding Twitter profiles, we identify Twitter-followers of the Twitter profiles of apps. Pseudo-documents are then created to represent users, where each pseudo-document contains the IDs of Twitter-followers of the apps that a user has previously downloaded. Thereafter, LDA is applied to the set of pseudo-documents to generate latent groups which is then used in the recommendation process. By combining the feature of Twitter-followers with other features based on various app metadata such as genre, developer, and the words in the app description, we can generate a much more accurate estimation of how likely a target user will like an app. Experimental results show that features extracted from Twitter consistently and significantly outperform state-of-the-art baselines which rely on (potentially misleading) textual information distilled from app descriptions. This also shows that *follower* information from Twitter helps us discover valuable signals that is effective in alleviating the cold-start situation.

In our future work, we plan to expand the use of data from SNS. For instance, second-degree relationships such as Twitter-followers *following* the current set of Twitter-followers may be useful, as would using data from other SNS, such as Facebook. Finally, we note that apps have a unique trait—they can be *updated*, which usually improves the app by incorporating new features or bug fixes. We plan to explore the use of this kind of temporal information to enhance our app recommender system.

7. ACKNOWLEDGEMENTS

This research is supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office.

8. REFERENCES

[1] C. C. Aggarwal, J. L. Wolf, K.-L. Wu, and P. S. Yu. Horting Hatches an Egg: A New Graph-theoretic Approach to Collaborative Filtering. In *Proc. of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'99)*, pages 201–212, 1999.

[2] R. Ayalew. *Consumer Behaviour in Apple's App Store*. PhD thesis, Uppsala University, 2011.

[3] E. Bakshy, J. M. Hofman, W. A. Mason, and D. J. Watts. Everyone's an Influencer: Quantifying Influence on Twitter. In *Proc. of the 4th ACM International Conference on Web Search and Data Mining (WSDM'11)*, pages 65–74, 2011.

[4] R. Bell, Y. Koren, and C. Volinsky. Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems. In *Proc. of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07)*, pages 95–104, 2007.

[5] D. M. Blei and J. D. Lafferty. Topic Models. *Text mining: Classification, Clustering, and Applications*, 10:71, 2009.

[6] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[7] J. Boyd-Graber, J. Chang, S. Gerrish, C. Wang, and D. M. Blei. Reading Tea Leaves: How Humans Interpret Topic Models. In *Proc. of the 23rd Annual Conference on Neural Information Processing Systems (NIPS'09)*, 2009.

[8] J. S. Breese, D. Heckerman, and C. Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proc. of the 14th Conference on Uncertainty in Artificial Intelligence (UAI'98)*, pages 43–52, 1998.

[9] M. Cha, H. Haddadi, F. Benevenuto, and K. P. Gummadi. Measuring User Influence in Twitter: The Million Follower Fallacy. In *Proc. of the 4th International AAAI Conference on Weblogs and Social Media (ICWSM'10)*, pages 10–17, 2010.

[10] J. H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29:1189–1232, 2001.

[11] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An Algorithmic Framework for Performing Collaborative Filtering. In *Proc. of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'99)*, pages 230–237, 1999.

[12] T. Hofmann. Latent Semantic Models for Collaborative Filtering. *ACM Transactions on Information Systems (TOIS)*, 22:89–115, 2004.

[13] T. Hofmann and J. Puzicha. Latent Class Models for Collaborative Filtering. In *Proc. of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 688–693, 1999.

[14] M. Jamali and M. Ester. A Matrix Factorization Technique with Trust Propagation for Recommendation in Social Networks. In *Proc. of the 4th ACM Conference on Recommender Systems (RecSys'10)*, pages 135–142, 2010.

[15] D. Kim and B.-J. Yum. Collaborative Filtering Based on Iterative Principal Component Analysis. *Expert Systems with Applications*, 28(4):823–830, 2005.

[16] Y. Koren. Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model. In *Proc. of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08)*, pages 426–434, 2008.

[17] Y. Koren and R. Bell. Advances in Collaborative Filtering. *Recommender Systems Handbook*, pages 145–186, 2011.

[18] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a Social Network or a News Media? In *Proc. of the 19th International Conference on World Wide Web (WWW'10)*, pages 591–600, 2010.

[19] N. N. Liu, X. Meng, C. Liu, and Q. Yang. Wisdom of the Better Few: Cold-Start Recommendation via Representative based Rating Elicitation. In *Proc. of the 5th ACM Conference on Recommender Systems (RecSys'11)*, pages 37–44, 2011.

[20] H. Misra, O. Cappé, and F. Yvon. Using LDA to Detect Semantically Incoherent Documents. In *Proc. of the 12th Conference on Computational Natural Language Learning (CoNLL'08)*, pages 41–48, 2008.

[21] Y. Moshfeghi, B. Piwowarski, and J. M. Jose. Handling Data Sparsity in Collaborative Filtering using Emotion and Semantic-based Features. In *Proc. of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'11)*, pages 625–634, 2011.

[22] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report SIDL-WP-1999-0120, Stanford Digital Library Technologies Project, 1998.

[23] S.-T. Park and W. Chu. Pairwise Preference Regression for Cold-Start Recommendation. In *Proc. of the 3rd ACM Conference on Recommender Systems (RecSys'09)*, pages 21–28, 2009.

[24] D. Ramage, S. Dumais, and D. Liebling. Characterizing Microblogs with Topic Models. In *Proc. of the 4th International AAAI Conference on Weblogs and Social Media (ICWSM'10)*, pages 130–137, 2010.

[25] R. Recuero, R. Araujo, and G. Zago. How Does Social Capital Affect Retweets. In *Proc. of the 5th International AAAI Conference on Weblogs and Social Media (ICWSM'11)*, pages 305–312, 2011.

[26] A. Said, E. W. De Luca, and S. Albayrak. How Social Relationships Affect User Similarities. In *Proc. of the 2010 Workshop on Social Recommender Systems*, pages 1–4, 2010.

[27] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted Boltzmann Machines for Collaborative Filtering. In *Proc. of the 24th International Conference on Machine Learning (ICML'07)*, pages 791–798, 2007.

[28] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based Collaborative Filtering Recommendation Algorithms. In *Proc. of the 10th International Conference on World Wide Web (WWW'01)*, pages 285–295, 2001.

[29] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and Metrics for Cold-Start Recommendations. In *Proc. of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'02)*, pages 253–260, 2002.

[30] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Matrix Factorization and Neighbor based Algorithms for the Netflix Prize Problem. In *Proc. of the 2nd ACM Conference on Recommender Systems (RecSys'08)*, pages 267–274, 2008.

[31] C. Wang and D. M. Blei. Collaborative Topic Modeling for Recommending Scientific Articles. In *Proc. of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'11)*, pages 448–456, 2011.

[32] J. Weng, E.-P. Lim, J. Jiang, and Q. He. TwitterRank: Finding Topic-sensitive Influential Twitterers. In *Proc. of the 3rd ACM International Conference on Web Search and Data Mining (WSDM'10)*, pages 261–270, 2010.

[33] S. Wu, J. M. Hofman, W. A. Mason, and D. J. Watts. Who Says What to Whom on Twitter. In *Proc. of the 20th International Conference on World Wide Web (WWW'11)*, pages 705–714, 2011.

[34] G.-R. Xue, C. Lin, Q. Yang, W. Xi, H.-J. Zeng, Y. Yu, and Z. Chen. Scalable Collaborative Filtering using Cluster-based Smoothing. In *Proc. of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'05)*, pages 114–121, 2005.

[35] K. Zhou, S.-H. Yang, and H. Zha. Functional Matrix Factorizations for Cold-Start Recommendation. In *Proc. of the 34th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'11)*, pages 315–324, 2011.