App Recommendation: A Contest between Satisfaction and Temptation

Peifeng Yin[†] Ping Luo[§] Wang-Chien Lee[‡] Min Wang[♯]

[†,‡]Department of Computer Science & Engineering, Pennsylvania State University
[§,♯]Hewlett Packard Labs China, Beijing, China
{[†]pzy102, [‡]wlee} @cse.psu.edu, {[§]ping.luo, [♯]min.wang6}@hp.com

ABSTRACT

Due to the huge and still rapidly growing number of mobile applications (apps), it becomes necessary to provide users an app recommendation service. Different from conventional item recommendation where the user interest is the primary factor, app recommendation also needs to consider factors that invoke a user to replace an old app (if she already has one) with a new app. In this work we propose an Actual-Tempting model that captures such factors in the decision process of mobile app adoption. The model assumes that each owned app has an *actual* satisfactory value and a new app under consideration has a *tempting* value. The former stands for the real satisfactory value the owned app brings to the user while the latter represents the estimated value the new app may seemingly have. We argue that the process of app adoption therefore is a contest between the owned apps' *actual values* and the candidate app's *tempting value*. Via the extensive experiments we show that the AT model performs significantly better than the conventional recommendation techniques such as collaborative filtering and content based recommendation. Furthermore, the best recommendation performance is achieved when the AT model is combined with them.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, Performance

Keywords

Smartphone Apps, App recommendation, Contest

1. INTRODUCTION

The popularity of smart phones accelerates the growth of mobile applications, or *apps* for short. As Apple reported, there have been over 500,000 apps available in iPhone's app store[1]. Moreover, there have been 25 million app downloads by March, 2012 [12]. With so many apps providing different functions (e.g., games, utility, books, etc.), it is difficult for users to find right apps they are looking for via keywords search [36]. Therefore, there is an urgent need to provide an effective app recommendation service.

Recommendation services have been deployed for many types of items such as books [18], music [7], movies [15] and even point-of-interests [38, 39]. Compared with those items, mobile apps have a unique characteristics, i.e., a downloaded app may affect the adoption of alternative apps. Most conventional items (e.g., books, movies, locations) are for *one-shot consumption*. For these products, users are typically ready or even glad to receive more new and similar items when they finish "consuming" old ones (e.g., reading books, watching movies, vising locations). Mobile apps are different. Once an app is downloaded, it serves the user continuously, thus referred to as *continuous consumption*, which may affect the user's decision of downloading some alternative apps[2]. For example, if a user has owned a weather forecast app, she is less likely to download another one that provides exactly the same function. Therefore, an app recommendation service may want to avoid such a recommendation.

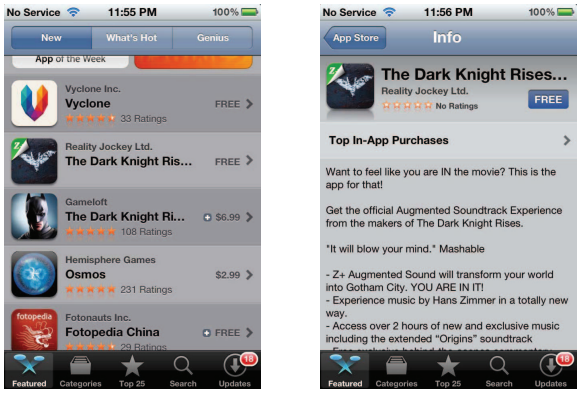
While avoiding recommendation of similar items for continuous assumption may sound intuitive, via an analysis of user downloading log, we observed that some users did download multiple similar apps (weather forecast, for example)[3]. Before detailing this we first consider a common process of app downloading. Usually a user enters an app store, which would provide her a list of apps, as shown in Figure 1a. Among those candidate apps, the user may pick one and tap the screen to view its detailed information, e.g., description, screenshots, reviews, etc., as shown in Figure 1b. If the user decides to download the app, she may click "install". If not, she may simply return to the app list. We summarize the process as *pick-and-take*, where "pick" represents the user's behavior of viewing the details of an app and "take" means the user downloads the app.

We argue that picking some app from a pool to view is affected by the users' interests since users can see the app titles while browsing the app list. On the other hand the decision

WSDM'13, February 4–8, 2013, Rome, Italy.

[1]http://www.apple.com/iphone/apps-for-iphone/
[2]We are aware that some apps are equal to electronic books and belong to one-shot consumption. However, most of them are to provide some utility which can continuously serve the user.
[3]Please refer to Section 5 for detailed description of the data.

to download the app is influenced by the contest between the already-owned apps and the candidate one. Specifically, people download apps to satisfy their needs. Thus, the apps' "satisfactory value", or marginal utility [33] is an important factor. However, such value remains unknown until the app is installed and used by the user. What prompting the user to download an app is its seeming attractiveness. Therefore, in this work we introduce the notions of *actual value* and *tempting value* to denote the assessment of satisfactory and attractiveness in mobile apps. The former refers to the real satisfactory value of the app after it is used and the latter stands for the estimated satisfactory value that the app seems to bring. Whether to download some app or not is thus a result of the contest between the owned apps' actual values and the candidate's tempting value.



(a) App list.

(b) App's detailed information page.

Figure 1: Screenshots of app store

Specifically, consider two apps, one owned by the user while the other is the candidate. There are basically two possible contest results between them. Firstly, if the candidate's tempting value is larger than the other's actual value, the user is more likely to download the new app. On the other hand, if the tempting value is smaller than the actual value, the user would probably not install the new app. For example, suppose the user has installed an app for weather forecast and now there is a new one available in the app store. If the app looks better than the owned one (e.g., more fancy screenshots), the user may probably download it. If the app does not seem to provide some attractive new features or powerful functions (i.e., the tempting value is no larger than the actual value), the user may not consider to install it.

In summary the process of an app download, as mentioned above, consists of two stages, namely pick to view and then download it. User interests and the contest between satisfaction and temptation are in play respectively in these two stages. Conventional recommendation techniques, which do not consider the latter one, may fail to provide successful recommendation when directly applied. We use examples in Figure 2 to illustrate scenarios where user-based collaborative filtering (UCF), item-based collaborative filtering (ICF) and content-based recommendation (CBR) fail to make good recommendation, respectively. UCF finds a set of users similar to the target user and recommends items accessed by those similar users to the target user. In Figure 2a, two users have shared a few apps, represented as *App Group*. UCF will thus treat them as similar users and recommend App_1 to u_2 and App_2 to u_1. However, if App_1's actual value is larger than App_2's tempting value, u_1 will reject the recommended App_2. Also, recommendation to u_2 will fail if the App_2 has an actual value larger than App_1's tempting value. Similarly for ICF and CBR, failing to incorporate the app contest results in ineffective recommendation.

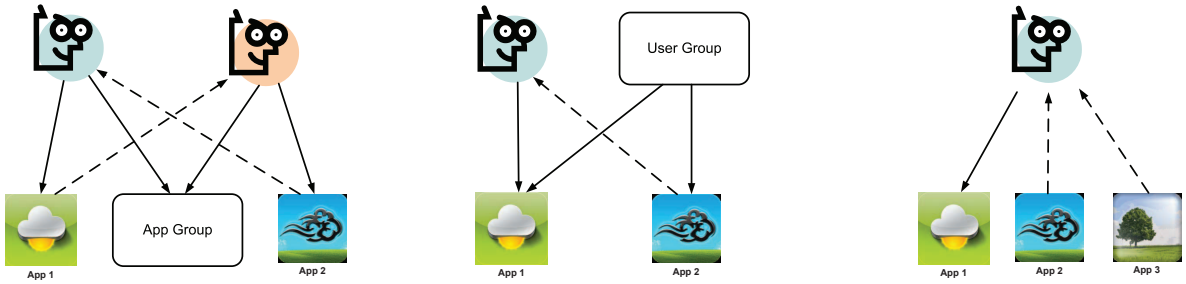
As shown above, the phenomenon of app contest is an important factor in app adoption and thus should be incorporated in app recommendation. In this work, we aim to model such phenomenon by mining the actual value (AV) and tempting value (TV) of each app. Generally they can be learned via the users' view/download sequences. With no loss of generality, consider two apps app_i and app_j, of which app_i is owned by the user and app_j is being viewed. There are two possible results, i.e., i) the user downloads app_j, and ii) the user does not download app_j. Let n_{ij} denote the number of people falling in case i) while m_{ij} denote the number of people in case ii). Intuitively, large n_{ij} and small m_{ij} indicate app_i's AV is larger than app_j's TV. Similarly, small n_{ij} and large m_{ij} suggest that the app_i's AV is smaller than app_j's TV. The details regarding learning model parameters are discussed in Section 3.2.

Usually, a user has owned several apps when she is viewing a candidate app. Intuitively not all of them may affect the user's decision. For example, a game app is unlikely to be selected for comparison when the user is considering whether to download a weather forecast app. Here we introduce a new metric *function overlap*, ranging from 0 to 1 to measure the similarity between two apps. The function overlap is 1 when two apps offer exactly the same function, e.g., weather forecast. On the other hand, function overlap is 0 if two apps are totally different, e.g., a game app and a weather forecast app. Generally, a larger function overlap between two apps suggests a higher probability of contest between them. In this work, while our focus is not on quantifying function overlap, we provide a reasonable measure based on each app's description. The details are provided in Section 2.2.

In summary, our contributions are four-folds:

- We reveal two item access patterns, i.e., one-shot consumption and continuous consumption. Also we point out a decision in app download is affected by two factors, i.e., user interest and app contest.
- We analyze why the state-of-the-art recommendation techniques, i.e., UCF, ICF and CBR, may fail in recommending apps due to app contest.
- We propose an Actual-Tempting model to capture the contest between apps and integrate it into app recommendation.
- We conduct extensive experiments to evaluate our AT model as well as collaborative filtering and content-based recommendation. The result shows that performance of conventional recommendation techniques is improved by incorporating the AT model.

The rest of the paper is organized as follows. Section 2 formulates the app recommendation problem and introduces the solution framework. It also briefly describes the quantification of function overlap. Section 3 provides details of our AT model. Section 4 gives a discussion on the AV and TV of each app. Section 5 evaluates the model as well as the baseline. Section 6 gives a literature review of related work. Finally Section 7 concludes the whole paper and briefly discusses our future work.



(a) User-based Collaborative Filtering (b) Item-based Collaborative Filtering (c) Content-based Recommendation

Figure 2: Three scenarios of failed recommendation. The solid arrow means the user downloads the app while the dashed arrow indicates the particular app is recommended to the user.

2. PRELIMINARIES

In this section we first formulate the app recommendation as a ranking problem. Then, we provide a simple method to measure the function overlap between two apps. Finally, we propose a ranking framework, which consists of two parts, i) user-interest ranking, and ii) app-contest ranking.

2.1 Problem Formulation

Generally, to recommend apps to a target user, the system selects a number, say k, apps from a pool of candidate apps and sort them based on the probability that the target user may download them. Formally let u_t denote the target user to whom the recommendation would be delivered, and $S = \{app_1, \cdots, app_n\}$ denote the list of candidates among which app_j is the j^{th} app in the list. The top-k app recommendation is to select k apps that are most likely to be downloaded by the target user u_t. The formal definition is shown in Definition 1.

DEFINITION 1 (TOP-k APP RECOMMENDATION). *Given a list of candidate apps S, the top-k recommendation is to form a subset S' s.t. $\forall app_{j_1} \in S', app_{j_2} \in S \wedge app_{j_2} \notin S'$, the user u_t is more likely to download app_{j_1} than app_{j_2}.*

2.2 Computation of Function Overlap

As mentioned in Section 1, the precondition that the user would make comparison of two apps is that these two apps should provide similar functions, or exactly, how much one app's function overlaps with the other's. While our focus is on the app contest, we only provide a feasible method to compute the function overlap. Formally, let $w_{ij} \in [0, 1]$ denote the function overlap between app_i and app_j, where small values indicate tiny function overlap and vice versa.

Supposedly an app's function is reflected in its description. Thus we use such information to indirectly quantify the function overlap. Specifically, we treat it as a document and use LDA [4] to learn the latent topic distribution for it. As such, each app is represented as a vector showing its distribution over the discovered latent topics. Then the function overlap of two apps is computed as the cosine similarity of their vectors. Let $\boldsymbol{f}$ denote the topic distribution of the app, the computation of function overlap between app_i and app_j is shown in Equation (1).

$$w_{ij} = \frac{\boldsymbol{f}_i \cdot \boldsymbol{f}_j}{||\boldsymbol{f}_i|| \times ||\boldsymbol{f}_j||} \quad (1)$$

where $(\cdot)$ represents the vector dot multiplication.

2.3 Ranking Framework

As described earlier the user behaviors in the mobile app market can be summarized as a *pick-and-take* process. First, when a user browses the app list she can see the titles and logos of the apps and thus know their basic functions. Thus, which apps are selected to view is mainly decided by the user's interests. Then, after she view the details of an app the contest between satisfaction and temptation plays a major role to decide to download it or not.

Therefore, in the scenario of app recommendation, there are two aspects to consider, user's interest and the app contest. The former suggests what kind of apps may attract the user while the latter considers the contest between the new app and old ones the user has already installed. Therefore, the ranking consists of two parts, user-interest ranking and app-contest ranking, which are linearly combined. Formally, the ranking score of an app app_i for the target user u_t is defined in Equation (2).

$$\texttt{Rank}(u_t, app_i) = \alpha \texttt{UIR}(u_t, app_i) + (1 - \alpha)\texttt{ACR}(u_t, app_i) \quad (2)$$

where $\alpha \in [0, 1]$ is a balancing parameter.

The first part $\texttt{UIR}(\cdot)$ measures how interesting the candidate app is, which can be implemented by any conventional recommendation technique. The second part $\texttt{ACR}(\cdot)$ consider the app contest and thus is the focus of this study.

3. ACTUAL-TEMPTING MODEL

In this section we describe the details of our AT model. We first introduce the general idea of the actual value and tempting value of each app as well as how they affect the user's app-downloading behaviors. Then we discuss how to learn such latent values for each app via users' action sequence. Finally we give the ranking function of AT model.

3.1 Model of Actual and Tempting Value

We assume that each app has two user-independent latent parameters, referred to as *actual value (AV)* and *tempting value (TV)*. The first one stands for the real satisfaction the app brings to the user while the second one is the estimated satisfaction the app may provide. The AV only exists after the user downloads and uses the app while the TV is obtained when the user views the app and evaluates whether it is worth downloading. Generally speaking, a large TV suggests that the app looks quite attractive and is therefore likely to motivate the user to download it.

Different apps may have functional overlap with each other,

Table 1: List of Symbols

u	The index over users
i, j	The index over apps
app_i	The i^{th} mobile app
a_i	The actual value of app_i
t_i	The tempting value of app_i
v_{ij}	The contest result of app_i and app_j
r_{uj}	The user u's action (download or not) to app_j
$A_{u,j}$	The collection of apps owned by u when viewing app_j
n_{ij}	The number of users who owned app_i and later download app_j
m_{ij}	The number of users who owned app_i and later does not download app_j

which becomes a factor the user may consider when deciding whether to download an app or not. For example, if the user already owns an app for weather forecast, then it is not very likely for her to download another one of the same function. However, a close check on the real data reveals that some users did download multiple apps of the same function, which can be naturally explained using the notion of AV-TV. Suppose there are two apps app_i and app_j serving the same function (e.g., weather forecast), the user already downloads app_i and is viewing the information of app_j. We argue that the process of decision-making involves the contest between the value of these two apps, or exactly, the AV of app_i and the TV of app_j. Specifically, if the app_i's AV is larger than the app_j's TV, the user may not download the app_i. On the other hand, if the app_i's AV is smaller than the app_j's TV, the user may be tempted to download the app_j since it is likely to bring her better user experience.

Formally, let a and t respectively denote the AV and TV of an app, p_{ij} stands for the probability that the user u owns app_i but downloads app_j later. We assume that the p_{ij} satisfies a Beta distribution whose parameters are app_i's a_i and AV and app_j's TV t_j, as shown in Equation (3).

$$P(p_{ij}|a_i, t_j) = \mathtt{Be}(p_{ij}; t_j, a_i) = \frac{\Gamma(a_i + t_j)}{\Gamma(a_i)\Gamma(t_j)}(p_{ij})^{t_j - 1}(1-p_{ij})^{a_i - 1} \tag{3}$$

where $\Gamma(x) = (x-1)!$ is a Gamma function.

Figure 3 shows the probability density function of p_{ij} for different combinations of (a_i, t_j). As is easily seen, for small a_i and large t_j, the generated probability p_{ij} is close to 1, indicating a high probability for the user to download app_j. On the other hand, if the a_i is large and t_j is small, the generated p_{ij} tends to be small, leading to a low probability of downloading app_j.

To model the app contest, we take two steps. Firstly, we give the probability of contest between two apps. Then we extend it to a general case, where contest occurs between one and a group of apps.

Contest between two apps.

Let $v_{ij} \in \{-1, 1\}$ denote the contest result between app_i and app_j, where -1 means the user does not download app_j (which loses the contest) and 1 means app_j is downloaded (wins the contest). Given two apps app_i and app_j, the v_{ij} satisfies a Bernoulli process whose parameter p is generated by a Beta distribution. Specifically the probability of v_{ij} is

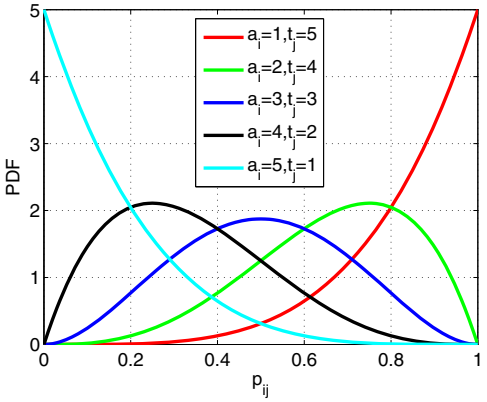


Figure 3: Probability density function of p_{ij} for different combination of (a_i, t_j).

shown in Equation (4).

$$\begin{aligned} P(v_{ij}|a_i, t_j) &= \int_0^1 \left(v_{ij}p_{ij} + \frac{1 - v_{ij}}{2}\right) \mathtt{Be}(p_{ij}; t_j, a_i) dp_{ij} \\ &= v_{ij} \int_0^1 p_{ij} \mathtt{Be}(p_{ij}; t_j, a_i) dp_{ij} + \frac{1 - v_{ij}}{2} \\ &= \frac{t_j}{t_j + a_i} \cdot v_{ij} + \frac{1 - v_{ij}}{2} = -\frac{a_i}{t_j + a_i} \cdot v_{ij} + \frac{1 + v_{ij}}{2} \end{aligned} \tag{4}$$

Contest between one and a group of apps.

Now we extend Equation (4) to a more general case. We assume that there exists a quantification method to compute the function overlap between any two apps. For two apps app_i and app_j, let $w_{ij} \in [0, 1]$ represent their function overlap. Suppose the user u now is viewing app_j and is considering whether to download it. At this time point, let $A_{u,j}$ denote the collection of apps u has already owned. Intuitively, the user would choose similar apps as reference to decide whether to download a new app. It is highly possible that she has installed multiple similar apps but we have no way to know which one affects her final decision. Thus we assume that for app contest, every owned app is likely to affect the final decision and the probability of choosing app_i for comparison is positively proportional to its function overlap with app_j.The probability of $r_{uj} \in \{1, -1\}$ can thus be written as in Equation (5).

$$P(r_{uj}|A_{u,j}, t_j) = \prod_{app_i \in A_{u,j}} P(v_{ij} = r_{uj}|a_i, t_j)^{\tau_{u,i,j}} \tag{5}$$

where $\tau_{u,i,j} \in \{1, 0\}$ is a binary value indicating whether the app_i is selected by the user u for comparison when she is viewing app_j and $P(\tau_{u,i,j} = 1) \propto w_{ij}$.

3.2 Model Learning

Here we describe the details of learning parameters for each app's AV and TV. The observed data is a series of logs sorted by the time stamp. Each log has such format as $(u, app_j, A_{u,j}, r_{uj})$, where u is the unique user id, app_j is the app the user is viewing, $A_{u,j}$ is the collection of apps already owned by the user and r_{uj} is the final action the user takes. When $r_{uj} = -1$, the user does not download

app_j and $r_{u,j} = 1$ means the user downloads the app. Let $\boldsymbol{A}$ and $\boldsymbol{T}$ denote the AV and TV for apps appearing in the data $\mathcal{D}$. The learning process is to find proper values for $\boldsymbol{A}$ and $\boldsymbol{T}$ such that the log-likelihood of observing $\mathcal{D}$ is maximized as shown in Equation (6).

$$\begin{aligned}\log P(\mathcal{D}|\boldsymbol{A},\boldsymbol{T}) &= \sum_u \sum_j \boldsymbol{I}_{uj} \log P(r_{uj}|A_{u,j},t_j) \\ &= \sum_u \sum_j \boldsymbol{I}_{uj} \sum_{app_i \in A_j^u} \tau_{u,i,j} \log P(v_{ij}|a_i,t_j)\end{aligned} \tag{6}$$

where $\boldsymbol{I}_{uj}$ is binary indicator where 1 means the user u has viewed (and even downloaded) app_j.

Since we do not really know which apps are selected for comparison when she views app_j, $\tau_{u,i,j}$ is unknown. Here we adopt the idea of Expectation-Maximization where the expected value of $\tau_{u,i,j}$ is computed and then the log-likelihood is maximized.

$$\begin{aligned}E_\tau(\log P(\mathcal{D}|\boldsymbol{A},\boldsymbol{T})) &= \sum_{u,j} \boldsymbol{I}_{uj} \sum_{app_i \in A_j^u} E(\tau_{u,i,j}) \log P(v_{ij}|a_i,t_j) \\ \propto \sum_u \sum_j \boldsymbol{I}_{uj} \sum_{app_i \in A_j^u} w_{ij} \log P(v_{ij}|a_i,t_j) &=_{DEF} \mathfrak{L}(\boldsymbol{A},\boldsymbol{T})\end{aligned} \tag{7}$$

We further transform Equation (7) by introducing two new values, n_{ij} and m_{ij}. The former one counts the number of people who own app_i and later download app_j while the latter one is the number of people who own app_i but do not download app_j after viewing it. Mathematically, these two values are defined in Equation (8).

$$\begin{aligned}n_{ij} &= \sum_u \boldsymbol{I}_{uj} \cdot \frac{1+r_{uj}}{2} \cdot \mathbf{1}_{app_i \in A_{u,j}} \\ m_{ij} &= \sum_u \boldsymbol{I}_{uj} \cdot \frac{1-r_{uj}}{2} \cdot \mathbf{1}_{app_i \in A_{u,j}}\end{aligned} \tag{8}$$

These two values can help remove the u from the objective function. Semantically, each log can be interpreted as one or multiple logs describing the contest between two apps. For example, given a log (u, app_j, A_j^u, v_j^u), we can unfold it to such set of data $S = \{(app_i, app_j, w_{ij} \cdot v_{ij} | app_i \in A_j^u)\}$. The learning is to find proper values for $\boldsymbol{A}$ and $\boldsymbol{T}$ such that the probability of these unfolded data is maximized. Now the objective function $\mathfrak{L}(\boldsymbol{A},\boldsymbol{T})$ can be written as Equation (9).

$$\begin{aligned}\mathfrak{L}(\boldsymbol{A},\boldsymbol{T}) &= \sum_j \sum_u \boldsymbol{I}_{uj} \sum_{app_i \in A_j^u} w_{ij} \log P(v_{ij}|a_i,t_j) \\ &= \sum_i \sum_{j \neq i} w_{ij} (n_{ij} \log P(1|t_j,a_i) + m_{ij} \log P(-1|t_j,a_i)) \\ &= \sum_i \sum_{j \neq i} w_{ij} \left(n_{ij} \log \frac{t_j}{t_j+a_i} + m_{ij} \log \frac{a_i}{a_i+t_j} \right)\end{aligned} \tag{9}$$

The parameters can be learned via gradient descent with the partial differentiation to each a_i and t_j, shown as in Equation (10).

$$\begin{aligned}\frac{\partial \mathfrak{L}}{\partial a_i} &= \sum_{j \neq i} w_{ij} \left(\frac{m_{ij}}{a_i} - \frac{n_{ij}+m_{ij}}{a_i+t_j} \right) \\ \frac{\partial \mathfrak{L}}{\partial t_j} &= \sum_{i \neq j} w_{ij} \left(\frac{n_{ij}}{t_j} - \frac{m_{ij}+n_{ij}}{t_j+a_i} \right)\end{aligned} \tag{10}$$

3.3 AT Ranking

With the learned actual value and tempting value for each app, recommendation can be made considering the possible contest result between new app and those owned by the user. As mentioned above, when the user is viewing an app, each of her owned apps is likely to affect the decision and the probability of influence is proportional to the function overlap between these two apps. The ranking score given by the AT model is the expected value of action r_{uj} where 1 for download and -1 for not.

Given two apps app_i and app_j, of which app_i is owned by the user and app_j is the one under review. With Equation (4) giving the probability, the expected value of v_{ij} can be computed in Equation (11).

$$E(v_{ij}|a_i,t_j) = 1 \cdot \frac{t_j}{a_i+t_j} + (-1) \cdot \frac{a_i}{a_i+t_j} = \frac{t_j-a_i}{a_i+t_j} \tag{11}$$

Given a set of apps $A_{u,j}$ owned by the target user u_t, the ranking score for app_j is the expected value for r_{uj}.

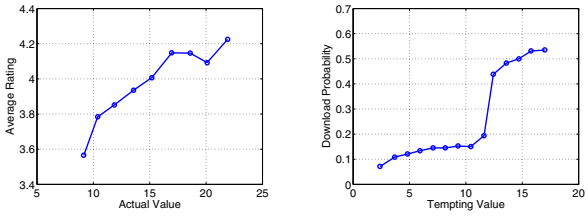
$$\begin{aligned}\texttt{AT}(u_t, app_j) &= E(r_{uj}|A_{u_t,j},t_j) \\ &\propto \sum_{app_i \in A_{u_t,j}} w_{ij} E(v_{ij}|a_i,t_j) = \sum_{app_i \in A_{u_t,j}} w_{ij} \cdot \frac{t_j-a_i}{a_i+t_j}\end{aligned} \tag{12}$$

4. ANALYSIS AND DISCUSSION

We use a data set containing 5,661 apps and 3,308 users' view-download record to learn our AT model. Details of the data set are described later in Section 5. Then we analyze the learned actual and tempting value for these apps. Generally, we aim to explore the following two apsects, i) the indication of actual and tempting values, ii) the hint of the difference between actual and tempting values.

4.1 The Indication of Actual and Tempting Values

The core idea of the AT model is that each app has two latent values, actual value and tempting value. The former is the real value the app may bring to the user once she downloads and uses it. The latter is the value estimated by the user when viewing it. In other words, the actual value reflects the real satisfaction of an app while the tempting value suggests the expectation of the app to the user. To demonstrate such claims, we crawled these apps' ratings from the app store and compare them with the apps' actual values. Also, we scanned the original records and computed for each app its number of downloads with regarding to the total number of viewing. Results are shown in Figure 4.



(a) Meaning of app's actual value (b) Meaning of app's tempting value

Figure 4: Meaning of app's actual and tempting value

Figure 4a displays the relationship between actual value and app rating in the app store. Among all apps, we firstly

find a minimum as well as maximum actual value. Next we equally divide this range into 9 groups and assign each app a group according to its learned actual value. Then for each group, we calculate the average ratings of apps. It can be seen that the two have a positive correlation where larger actual value suggests a higher rating. Since the app's rating in app store reflects the app's quality in some way, we show that the actual value of our AT model captures the app's quality to some degree.

Figure 4b displays the relationship between tempting value and app's download probability. Similar to the above steps, we group the apps into 14 bins based on their learned tempting values. Then for each group, we calculate the average proportion of download numbers over view numbers, referred to as *download probability*. It can be easily seen that with the growth of tempting value, the download probability is also increasing. Given that the user has no idea of the app's real quality before downloading it, the tempting value thus suggests the degree of attraction the app looks.

4.2 The Hint of Actual-Tempting Difference

After a careful check on the learned model, we note that there exists apps whose actual value and tempting value are not equal, and the difference may vary from app to app. Thus we are curious about the hint behind such difference. In Apple's app store, each app is assigned a category. Accordingly, we group apps and for each category calculate the average difference between its actual value and tempting value. The results are shown in Figure 5.

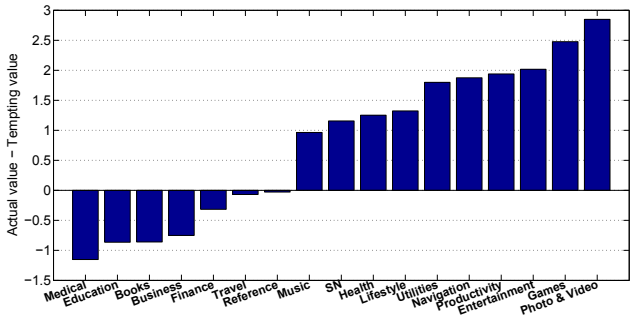


Figure 5: Actual-tempting difference with regarding to app category. Note that negative value means the app's actual value is smaller than its tempting value and vice versa.

Note that for apps in the same category, negative actual-tempting difference may lead to a continuous downloading of similar apps. The most straightforward example is the category Books. In app store, apps of this category are usually some electric books or comics, e.g., G. A. Henty Books[4], Pocket God Comics[5] etc. These apps are similar to conventional items (e.g., real books) where the consumption is one-shot, i.e., users are ready to receive new items once finishing old ones.

Another interesting app category is those whose actual value and tempting value are quite close. Similar actual and tempting value means the app's real value is easy to estimate and one knows what to expect before downloading. The representative example is the category Reference. Apps in this category usually provide guidance or instructions, e.g., MW3 Facts and Guide[6], Encyclopedia of Guns[7]. Users clearly know these apps' function before downloading. Therefore the difference between actual and tempting values is nearly zero.

Finally some apps have higher actual values than tempting values. Apps falling in this type are more likely to block users from downloading other similar apps since on average their actual values are larger than tempting values. In reality, these apps usually have strong function overlap and downloading similar ones is unnecessary. For example, one is less likely to download two radio players in Music category. Also, in Social Networking (SN), once a user is using one (e.g., Facebook), she is unwilling to switch to another (e.g., Google+).

5. EVALUATION

In this section we conduct extensive experiments to demonstrate the effectiveness of our AT model as well as other recommendation techniques. We also evaluate the performance of hybrid recommendation that integrates AT model into collaborative filtering and the content-based recommendation techniques, respectively.

5.1 Data Preprocess

We used the data logged from an iPhone app, namely, *Limited-time Free*[8], which collects information of iPhone apps that become available for free for a limited time in the Apple App Store. When a user runs the app, she is given a list of apps that are free for a limited time, as shown in Figure 6a. When the user clicks an app from the list, the detailed description as well as its running screenshots are displayed (Figure 6b). When the user decides to download the app, she clicks the "Get" button, which leads her to the app store. Also, if she does not want to download the app, the user can return to the app list by clicking the "Back" button. Compared with the official Apple App Store as shown in Figure 1, the user interface of this software is very much similar and thus we can safely claim that users follow the same *pick-and-take* pattern when downloading apps via this one. The system records these two types of actions for each user, i.e., the user's picking some particular app to view detailed information and her downloading some app. By analyzing each log entry, we know which apps the user has downloaded and which apps the user viewed only (but has not downloaded). We use the afore-described data for evaluation.

We further process the log as follows. We remove users who have conducted less than 100 actions and then remove apps that have less than 10 visitors. The resulted data set consists of 3,308 users and 5,661 apps. Among these users, we randomly select 25% as target users to receive recommendation. For each target user, we mask their most recent 25% log entries by default. Recommendation is evaluated by observing how many masked logs are recovered in the recommendation list.

5.2 Evaluation Metrics

In the experiment, we use "relative" *precision* and *recall* to measure the recommendation performance of the recommen-

[4]http://itunes.apple.com/app/id397900901
[5]http://itunes.apple.com/app/id380752345
[6]http://itunes.apple.com/app/id479406467
[7]http://itunes.apple.com/app/id368360223
[8]http://itunes.apple.com/app/id440230030

(a) Screenshot of app list. (b) Screenshot of app's detailed information page.

Figure 6: Screenshots of Limited-free app

dation technique under examination improves over a random recommender. Given a recommendation list l sorted in descending order of the ranking score, we refer to those that are previously masked from the log as "hit". Let $hit(k)$ denote the number of hit in a top-k list, and H is the total number of masked items. The absolute precision and recall are defined in Equation (13) as follows.

$$precision@k = \frac{hit(k)}{k} \qquad recall@k = \frac{hit(k)}{H} \tag{13}$$

Let S denote the total number of candidate items, the precision and recall in a top-k list of a random recommender system is $\frac{H}{S}$ and $\frac{k}{S}$ [41]. Therefore, the relative precision and recall is defined in Equation (14).

$$\begin{aligned} \text{r-}precision@k &= \frac{precision@k}{H/S} = \frac{hit(k)\cdot S}{kH} \\ \text{r-}recall@k &= \frac{recall@k}{k/S} = \frac{hit(k)\cdot S}{kH} \end{aligned} \tag{14}$$

As shown, the relative precision and recall are the same and thus we only show one of them in the experiment result. Furthermore, the groundtruth data contains two types of items (apps), i) those viewed and downloaded by the user, and ii) those viewed only by the user. Naturally an ideal recommendation would rank items in first group as high as possible and rank those in second group as low as possible. Accordingly we evaluate these two types of items separately. In the experiment we first sort the recommended apps in a descent order based on the ranking score and count how many of top-k ranked apps are really downloaded by the target user in the masked ground truth data. This performance is denoted as P_{pos}. Then we resort the list in an ascending order and count how many of top-k ranked apps are only viewed but not downloaded by the target user, denoted as P_{neg}. For both metrics, higher value indicates a better recommendation performance.

5.3 Baseline

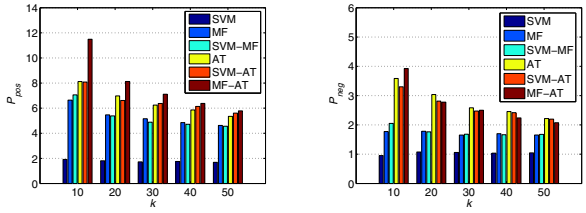
In the evaluation, we mainly consider two conventional recommendation techniques, i.e., collaborative filtering (CF) and content-based recommendation (CBR). Specifically we use probabilistic matrix factorization (PMF) [22] to realize CF and support vector machine (SVM), implemented by LIBSVM [5] to realize CBR. PMF has been widely used in previous works [23, 1, 17, 2] as an implementation of collaborative filtering. It is highly flexible and easy to extend. SVM displays good performance in classification and its probabilistic extension [20, 40] can be used for ranking.

As mentioned above, apps viewed by the user are either downloaded or not. In the user-item matrix, we assign 1 to the element if the corresponding app is downloaded by the user and set the value as -1 if the app is only viewed but not downloaded by the user. For SVM, we use the description of the app as the feature vector. Also, apps are labeled as positive example if they are downloaded and those that are only viewed are labeled as negative example.

Besides pure PMF and SVM, we also displays performance of hybrid methods that linearly combine any of two solutions, including PMF & SVM, PMF & AT and SVM & AT.

5.4 General Performance

We firstly evaluate the general performance of all recommendation techniques. For hybrid ones, we try different weighting parameters and display the result of the optimal one. The results are shown in Figure 7.



(a) P_{pos} (b) P_{neg}

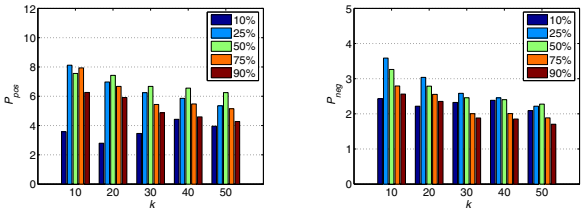
Figure 7: General performance of examined techniques.

It can be easily seen that the AT model as well as solutions that have incorporated the AT model outperform others. Note that the AT model takes into consideration the effect of contest between apps when making recommendation. Its better performance confirms our argument that in mobile app recommendation, the app contest is an important factor to be considered. Also, those hybrid methods that incorporates the AT model, i.e., SVM-AT and MF-AT, achieve better performance than the pure AT model. This is because AT, emphasizing on unique relationships between apps, are complementary to other techniques that capture users' interests. Therefore optimal performance is obtained when these techniques are integrated together. Finally, SVM performs worse than MF in terms of P_{pos}, but outperforms MF in terms of P_{neg}.

The former observation indicates that the impact of app contest is more significant to content-based recommendation than to collaborative filtering. This can be explained as follows. Recall that in Figure 2c we show how content-based recommendation may fail in app recommendation. Its idea of recommending items similar to those appeared in a user's historical log is susceptible to the app contest. The latter one is caused by imbalanced classification problem [6, 34]. In the records, the proportion of download log is relatively small compared to view log, because users would usually view multiple apps before downloading one. This situation leads to an imbalanced training data set for SVM which is biased towards a strong class (i.e., apps that are viewed only by the user) when conducting classification.

5.5 Experiment on Mask Ratio

In this set of experiment we evaluate the performance of our AT model with regarding to different size of test data, i.e., mask ratio. After randomly selecting a group of target users, we mask a proportion of their most recent app view/download log and check how many of them are recovered by the AT model. The number of masked log is mask ratio. The default value is 25%. In this experiment adjust this ratio to different values, i.e., 10%, 50%, 75% and 90%, where large value indicates small training data. Results are shown in Figure 8.



(a) P_{pos} (b) P_{neg}

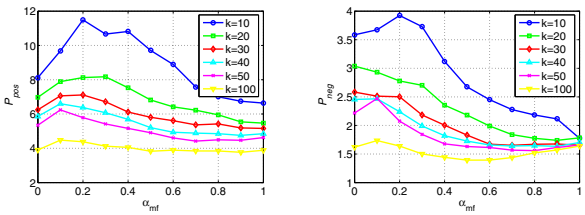
Figure 8: Experiment on mask ratio

From the figures we can see that when the mask ratio increases from 25% to 90%, the performance degrades. This is reasonable as the training data is decreasing, resulting in an inaccurate learned model. Therefore the recommendation precision suffers. On the other hand, when ratio increases from 10% to 25%, the performance is improving due to the increase of test data. When the mask ratio is 10%, the proportion of masked apps in the recommendation pool is very tiny which makes the recommendation quite difficult to hit the masked ones. Therefore the performance is poor. When the test data is properly increased, the performance improvement is be expected.

5.6 Experiment on Balancing Parameters

In this experiment we demonstrate how much improvement the AT model brings to collaborative filtering and content-based recommendation with different balancing parameters. Recall in Equation (2) the ranking score is the linear combination of two. In the experiment, we combine the AT model respectively with MF and SVM, where the balancing parameters are denoted as α_{mf} and α_{svm}.

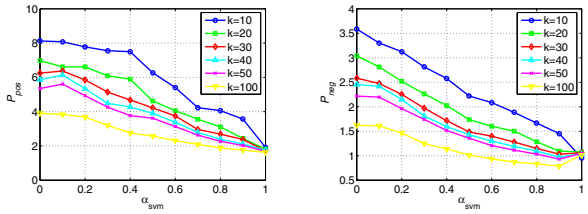
The balancing parameter α_{mf} controls the weight of collaborative filtering in the final ranking score. When α_{mf} is set to 0, the recommendation is reduced to pure AT while $\alpha_{mf} = 1$ transforms the model to the pure CF. Figure 9 shows the impact of α_{mf} to the recommendation performance, where optimal one is achieved when $\alpha_{mf} = 0.2$.



(a) P_{pos} (b) P_{neg}

Figure 9: Experiment on α_{mf}

Next we test the hybrid of content-based recommendation and AT model with regarding to different values for α_{svm}. Note that when $\alpha_{svm} = 0$, the hybrid one is reduced to AT model. And when $\alpha_{svm} = 1$, the solution is equal to content-based recommendation. As shown in Figure 10, the performance is improved when two methods are combined and the optimal performance is achieved when $\alpha_{svm} = 0.1$. Also, recall that when CF is combined with AT, the optimal balance weight is 0.2. The smaller weight for CBR indicates that the contest effect has a larger impact on its performance.

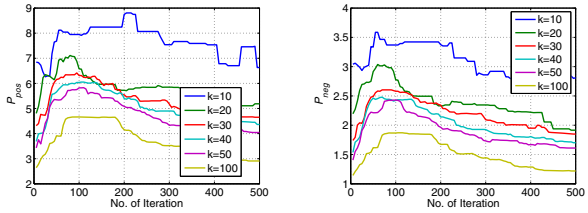


(a) P_{pos} (b) P_{neg}

Figure 10: Experiment on α_{svm}

5.7 Experiment on No. of Iterations

In the final set of experiments, we evaluate the performance of the AT model with regarding to different number of iterations. In gradient descent, we set the learning speed as 0.001. The results are shown in Figure 11.



(a) P_{pos} (b) P_{neg}

Figure 11: Experiment on different No. of iterations

As can be seen in Figure 11a and Figure 11b, the performance of the AT model is optimal near the 60th iteration in terms of both P_{pos} and P_{neg}. When the iteration number exceeds 60, the performance is degrades, indicating the model is over-fit.

6. RELATED WORK

Due to the popularity of smart phones and growing number of mobile app made available, a number of research works on the mobile app recommendation have appeared in the literature [35, 36, 9, 11]. However, most of them focus on how to collect extra information via the phones in order to help improve recommendation. In [35] several app usage patterns have been identified. In [36, 9], app's usage data is collected to quantify user's rating on downloaded apps. In [11], GPS sensor information is exploited to provide context-aware app recommendation. Those works are complementary to ours in terms of app recommendation. In this work, we mainly focus on a unique phenomenon in mobile

app adoption, i.e., *app contest*, which makes mobile app recommendation different from conventional item recommendation where current recommendation techniques, e.g., CF and CBR, are directly applicable.

Conventional recommendation techniques are generally classified into three categories, collaborative filtering, content-based recommendation and hybrid of the two. In the following we give a brief review of the first two as well as their implementation techniques. Finally we review recent works that consider item relationship in recommendation.

6.1 Collaborative Filtering and Probabilistic Matrix Factorization

Collaborative filtering (CF) is a well known recommendation approach that has been widely studied and adopted. The core idea is that users of similar interests may display similar ratings on similar items. Specifically, if a recommendation is made based on similar users (those who have accessed the same items as the target user has), it is referred to as *user-based CF* [13]. On the other hand, if a recommendation is made based on similar items (those that have been accessed by the same set of people), it is referred to as *item-based CF* [25].

As mentioned in [10], CF algorithms can be roughly classified into neighborhood models and latent factor models. Matrix factorization (MF) is one of the most successful realizations of latent factor models and is superior to classic nearest-neighbor techniques [14]. The general idea of MF is that users and items are associated with several latent factors and the observed user-item rating matrix is the multiplication of the two. These latent factors are learned by minimizing a loss function, which is defined as the sum-squared distance between observed and simulated entries [28]. Then Rennie et. al. [29, 21] modified such a loss function by constraining the norms of user and item feature vectors.

In [22], Salakhutdinov and Mnih proposed probability distribution on the latent feature vectors and developed *probabilistic matrix factorization (PMF)*. In their work, they derived the loss function in terms of probability theory and correlated those heuristically-defined loss function with standard Gaussian distribution. Based on the idea of PMF, many variants and extensions have been proposed so far, e.g., [23, 1, 17, 2]. Readers can resort to [26, 27] for a general picture.

In this work we adopt the basic PMF proposed in [22] as the implementation of CF. Our goal is to demonstrate the importance of app contest in app recommendation and performance would be improved when combined with our AT model.

6.2 Content-based Recommendation and Support Vector Machine

The content-based recommendation (CBR) is an outgrowth and continuation of information filtering research [3]. It recommends items similar to the target user's profile. A typical CBR consists of three steps, i.e., content analyzer (or item representation), user profile learning and content filtering [16]. The first one is about how to model items' feature. The second and third one are usually bound together.

For textual items, the feature modeling has been widely studied for information retrieval where items are usually represented as bag-of-word with TF/IDF [24] or latent topic distribution [30]. The latter one is proved to be more precise and thus in this work we use the popular Latent Dirichlet Analysis (LDA) [4] to model the apps' features.

Major methods of user-profile learning, as mentioned in [19, 16], are k-nearest neighbor, Naive Bayes classifier and linear/non-linear classifier. The first one is inefficient at classification time [16]. The second one suffers unsatisfactory performance for diversified length of items' textual content and unbalanced training sample [16]. Therefore in this work we choose Support Vector Machine (SVM) [32] as it is shown to be a good classifier in text categorization [31] and has been adopted by previous works (e.g., [40, 8]) as basis of CBR. Furthermore, the probabilistic extension [20, 40] makes it capable of ranking tasks as in our work.

6.3 Item Relationship in Recommendation

Few works considered the impact of item relationship in recommendation except for two recent works [37, 33], where the relationship is modeled by introducing the concept of "utility".

In [37], Yang et. al. proposed *Collaborative Competitive Filtering (CCF)* to model the user choice process. They argued that when the user is given a few items, they are competing to win the user's favor. Our work is different. Firstly, in [37] they model the item competition among candidate items, but in this work we aim to model the contest between old items (apps) and new one. Also, the scenario is different. In [37], user first searched for similar items and selected only one of them. In our case, the recommended apps are not necessarily similar. Even if they are, the user can select (download) all of them since the cost is not high[9].

In [33], Wang et. al. modeled the user purchase behavior in E-commerce by maximizing the marginal net utility. Specifically they proposed a utility function where each item has decreasing benefit as the purchasing quantity increases, based on the Law of Diminishing Marginal Utility. The app recommendation in our work differs from conventional shopping behavior in two ways. Firstly, there is no concept of "quantity" in app downloading. The user does not need to repetitively download one app as she might do when purchasing some conventional items, e.g., food, tickets. For app, there is only binary value, i.e., download or not. Secondly, in [33], the increase of some item can always bring some utility, although the value is decreasing as the quantity increases, which is not true in our case. For example, if the user already has a weather forecast app and suppose it works perfectly well, downloading any similar app brings nothing but occupance of smart phone's storage space. Thus, the assumption in [33] is not applicable to app recommendation.

7. CONCLUSION AND FUTURE WORK

In this work we proposed the AT model to exploit the app contest phenomenon for app recommendation. Generally we assume each app has two latent values, i.e., *actual value* and *tempting value*. The former stands for the real value the app brings to the user while the latter represents the satisfactory value it looks to have. The app recommendation needs to consider both user's interest and the app contest. App download involves a comparison between the actual satisfactory values of owned apps and the tempting value of a new app. By conducting extensive experiments, we demonstrate that our AT model outperforms conventional recommendation techniques (i.e., CF and CBR) that only captures user's

[9]All apps we studied in this work are free when the user views them

interests. Moreover, optimal recommendation is achieved when both parts are integrated.

In the proposed model, the actual and tempting values are treated as latent attributes of apps and learned via users' action sequence. While this technique is quite popular in latent factor model, it is rather interesting to understand what really determines the actual and tempting values for an app. In future works we plan to conduct a comprehensive study on this point.

8. REFERENCES

[1] D. Agarwal and B.-C. Chen. flda: matrix factorization through latent dirichlet allocation. In *WSDM*, pages 91–100, 2010.

[2] N. Barbieri and G. Manco. An analysis of probabilistic methods for top-n recommendation in collaborative filtering. In *ECML/PKDD (1)*, pages 172–187, 2011.

[3] N. J. Belkin and W. B. Croft. Information filtering and information retrieval: two sides of the same coin? *Commun. ACM*, 35(12):29–38, 1992.

[4] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003.

[5] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[6] N. V. Chawla, N. Japkowicz, and A. Kotcz. Editorial: special issue on learning from imbalanced data sets. *SIGKDD Explor. Newsl.*, 6(1):1–6, 2004.

[7] H.-C. Chen and A. L. P. Chen. A music recommendation system based on music data grouping and user interests. In *CIKM*, pages 231–238, 2001.

[8] K.-W. Cheung, J. T. Kwok, M. H. Law, and K.-C. Tsui. Mining customer product ratings for personalized marketing. *Decis. Support Syst.*, 35(2):231–243, 2003.

[9] E. Costa-Montenegro, A. B. Barragáns-Martínez, and M. Rey-López. Which app? a recommender system of applications in markets: Implementation of the service for monitoring users' interaction. *Expert Syst. Appl.*, 39(10):9367–9375, 2012.

[10] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *RecSys*, pages 39–46, 2010.

[11] C. Davidsson and S. Moritz. Utilizing implicit feedback and context to recommend mobile applications from first use. In *CaRR*, pages 19–22, 2011.

[12] D. Graziano. Apple's app store reaches 25 billion downloads. *http://www.bgr.com/2012/03/05/apples-app-store-reaches-25-billion-downloads/*, 2012.

[13] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *SIGIR*, pages 230–237, 1999.

[14] Y. Koren, R. M. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[15] G. Lekakos and P. Caravelas. A hybrid approach for movie recommendation. *Multimedia Tools Appl.*, 36(1-2):55–70, 2008.

[16] P. Lops, M. de Gemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender Systems Handbook*, pages 73–105. 2011.

[17] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King. Recommender systems with social regularization. In *WSDM*, pages 287–296, 2011.

[18] R. J. Mooney and L. Roy. Content-based book recommending using learning for text categorization. In *DL*, pages 195–204, 2000.

[19] M. J. Pazzani and D. Billsus. Content-based recommendation systems. In *The Adaptive Web: Methos and Strategies of Web Personalization*, pages 325–341. Springer-Verlag, 2007.

[20] J. C. Platt. *Probabilities for SV Machines*, pages 61–74. MIT Press, 2000.

[21] J. D. M. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *ICML*, pages 713–719, 2005.

[22] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *NIPS*, 2007.

[23] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *ICML*, pages 880–887, 2008.

[24] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1989.

[25] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295, 2001.

[26] H. Shan and A. Banerjee. Generalized probabilistic matrix factorizations for collaborative filtering. Technical report, University of Minnesota, Twin Cities, 2010.

[27] H. Shan and A. Banerjee. Generalized probabilistic matrix factorizations for collaborative filtering. In *ICDM*, pages 1025–1030, 2010.

[28] N. Srebro and T. Jaakkola. Weighted low-rank approximations. In *ICML*, pages 720–727, 2003.

[29] N. Srebro, J. D. M. Rennie, and T. Jaakkola. Maximum-margin matrix factorization. In *NIPS*, 2004.

[30] M. Steyvers and T. Griffiths. *Handbook of Latent Semantic Analysis*, chapter Probabilistic Topic Models.

[31] J. T. Text categorization with support vector machines : Learning with many relevant features. *ECML*, pages 137–142, 1998.

[32] V. N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., 1995.

[33] J. Wang and Y. Zhang. Utilizing marginal net utility for recommendation in e-commerce. In *SIGIR*, pages 1003–1012, 2011.

[34] G. M. Weiss. Mining with rarity: a unifying framework. *SIGKDD Explor. Newsl.*, 6(1):7–19, 2004.

[35] Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman. Identifying diverse usage behaviors of smartphone apps. In *IMC*, pages 329–344, 2011.

[36] B. Yan and G. Chen. Appjoy: personalized mobile application discovery. In *MobiSys*, pages 113–126, 2011.

[37] S.-H. Yang, B. Long, A. J. Smola, H. Zha, and Z. Zheng. Collaborative competitive filtering: learning recommender using context of user choice. In *SIGIR*, pages 295–304, 2011.

[38] M. Ye, P. Yin, and W.-C. Lee. Location recommendation for location-based social networks. In *GIS*, pages 458–461, 2010.

[39] M. Ye, P. Yin, W.-C. Lee, and D. L. Lee. Exploiting geographical influence for collaborative point-of-interest recommendation. In *SIGIR*, pages 325–334, 2011.

[40] K. Yu, A. Schwaighofer, and V. Tresp. Collaborative ensemble learning: combining collaborative and content-based information filtering via hierarchical bayes. In *UAI*, pages 616–623, 2003.

[41] T. Zhou, Z. Kuscsik, J.-G. Liu, M. Medo, J. R. Wakeling, and Y.-C. Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107:4511–4515, 2010.