AppFunnel: A Framework for Usage-centric Evaluation of Recommender Systems that Suggest Mobile Applications

Matthias Böhmer
DFKI GmbH
Saarbrücken, Germany
matthias.boehmer@dfki.de

Lyubomir Ganev
Saarland University
Saarbrücken, Germany
luboganev@gmail.com

Antonio Krüger
DFKI GmbH
Saarbrücken, Germany
antonio.krueger@dfki.de

ABSTRACT

Mobile phones have evolved from communication to multi-purpose devices that assist people with applications in various contexts and tasks. The size of the mobile ecosystem is steadily growing and new applications become available every day. This increasing number of applications makes it difficult for end-users to find good applications. Recommender systems suggesting mobile applications are being built to help people to find valuable applications. Since the nature of mobile applications differs from classical items to be recommended (e.g. books, movies, other goods), not only can new approaches for recommendation be developed, but also new paradigms for evaluating performance of recommender systems are advisable. During the lifecycle of mobile applications, different events can be observed that provide insights into users' engagement with particular applications. This gives rise to new approaches for evaluation of recommender systems. In this paper, we present *AppFunnel*: a framework that allows for usage-centric evaluation considering different stages of application engagement. We present a case study and discuss capabilities for evaluating recommender engines by applying metrics to the *AppFunnel*.

ACM Classification Keywords
H.3.3 Information Search and Retrieval: Relevance feedback

Author Keywords
Recommendation; mobile apps; usage-centric evaluation.

INTRODUCTION

In recent years mobile phones have evolved from single-purpose to multi-purpose devices. In particular, this was facilitated by improved hardware and more effective programming capabilities. While earlier mobile phones provided only fixed and limited functionality, i.e. phone calls and short messaging services, current smartphones can be customized by their owners to extend functionality with new applications, also called *apps*. The advent of mobile app stores eased the process of distributing and installing mobile apps. This resulted in apps of various types and quality being made available for end-users. The number of app downloads is anticipated to surpass 81.4 billion in 2013.[1] In January 2013, there are more than 700,000 apps available for both Apple's iPhone and for the Android platform.[2] The number of available apps is steadily increasing. This results in end-users being overwhelmed by quantity, and finding good apps can become a difficult task.

Given this problem and the fast growth of the mobile ecosystem, recommendation of mobile apps became a research topic of increasing interest. Current approaches mainly focus on social aspects (e.g. [9, 1]), context awareness (e.g. [26, 21, 13, 8]) and characteristics of app markets (e.g. [20]). Exploiting context seems especially promising since smartphone usage is subject to perpetually changing contexts [12, 6, 24]. As such, the set of required apps also varies according to the user's changing tasks, e.g. from picture taking, to navigating, to looking for a restaurant. Mobile apps are a special type of items to be recommended: their usage is not only conditioned by users' contexts, but also the usage of a mobile app itself can be tracked alongside the user's interaction with the app — due to an app being a software process that can be observed [6]. Most importantly smartphone use can be characterized by "dead" apps: fewer than half of installed apps are actually being used [22]. Therefore, choosing installations as the evaluation metric might result in even more apps residing unused on a user's device. Consequently, new approaches for evaluating app recommender systems are required: They need to go beyond ratings, click-through-rates or download statistics. Hence, this paper gives rise to new paradigms for evaluating mobile app recommender systems.

Our two key contributions are: (1) We introduce a usage-centric approach for observing people's engagement with mobile apps on different stages along apps' lifecycles beyond installations. Therefore, we adopt the concept of conversion funnels to the mobile ecosystem and present *AppFunnel*. Based on that, we can track the performance of different recommender engines according to this funnel. (2) We present results of a case study of *AppFunnel*.

[1] `http://www.gartner.com/it/page.jsp?id=2153215`
[2] List of digital distribution platforms `http://is.gd/pzjWb6`

This shows capabilities for ascertaining the performance of different recommender engines by applying metrics to the *AppFunnel*. Further, we discuss preliminary findings showing that a context-aware recommender engine performs better when aimed on short-term usage at the time of the recommendation request, and a non-context-aware engine performs better when addressing long-term usage of apps.

RELATED WORK

Work that relates to ours covers mobile app recommendation, conversion funnels and user-centric evaluation.

Recommendation of Mobile Apps

Woerndl et al. [25] present the first recommender system suggesting mobile applications. It exploits context features and is based on capturing the installations of applications related to the context (basically location). The recommender engine is based on a hybrid engine following a multi-dimensional approach. Jannach and Hegelich [11] evaluate recommender engines suggesting game applications on a mobile internet platform. They found that the personalization of recommendations results in an increased number of views and sales. Yan and Chen [26] present the *AppJoy* system: a recommender system that suggests mobile applications on the *Google Play Market*. It is based on application usage data that is modeled according to the recency, frequency and duration of the application usage. Yan and Chen show that their recommendation approach based on implicit usage data performs well without explicit user input; they evaluated their system by comparing usage times of recommended and not-recommended apps. Davidsson and Moritz [8] present the *Applause* system, which is a recommender system that exploits users' locations to recommend apps. In particular *Applause* tends to exploit context to be able to recommend applications instantly without collecting any additional user data. Girardello et al. [9] present *AppAware*: an app suggesting recommender system that is based on people's application installations, uninstallations and updates. The system recommends applications based on their popularity, i.e. how many times an app was installed without removal. Böhmer et al. [5] present the prototype of a recommender system for mobile applications called *appazaar*[3]. We used this system as a testbed for the research presented in this paper. In [6] Böhmer et al. present a large-scale study on mobile application usage. Therefore they define an app's lifecycle to consist of installation, open, close, update, and removal events. Their findings can be exploited for context-aware recommendation of apps, e.g. news apps are more popular in the morning, and games at night.

Context-aware models for recommendation of mobile applications have recently been presented by Karatzoglou et al. [13] and Shi et al. [21]. Both papers present tensor-factorization based models that perform better than state of the art context-aware approaches and non-context-aware approaches when using app usage as implicit feedback. Also, Shi and Ali [20] present a recommender engine that addresses characteristics of mobile app markets. Their model is based on app usage as implicit feedback and specially tailored towards mobile app markets that have heavier heads and longer tails in their distributions.

However, none of the related works on mobile app recommendation uses a usage-centric model based on how engaged a user is with an application for evaluation. This is what this paper aims to do.

Conversion Funnels

A *conversion funnel* describes an action sequence, e.g. from showing an ad to a user, to the user clicking on the ad, to the user eventually purchasing the advertised product. The paradigm of *conversion funnels* is mostly used in the domains of marketing and advertising. Bagherjeiran et al. [2] present a ranking method for scheduling ads that is optimized for different stages of the conversion funnel. Their model enables a more efficient targeting of ads towards conversions. For online advertising, Becker et al. [4] provide insight into the relationship between landing page types, query classes, and conversion. They conclude that the majority of ad landing pages fall into three distinct classes: home pages, sub pages, and search result pages. Furthermore, the authors correlate these classes with conversion data provided by advertisers to show that the *conversion rate* (i.e., the proportion of certain users who take a predefined, desirable action, such as a purchase, registration, download, etc., as compared to simply page browsing) varies considerably according to these classes. Rosales et al. [18] provide a detailed analysis of conversion rates in the context of non-guaranteed delivery for display advertising. They formalize the problem of predicting the post-click conversion, i.e. conversions after a user clicks on a referring ad. The authors further provide fundamental properties of the post-click conversion process based on contextual information including a comparison between click-through rate and conversion rate.

In this paper, we apply the concept of conversion funnels to the domain of mobile application recommendation. The nature of mobile apps — being pieces of software that can be observed during an app's lifecycle [6] — allows us to define an action sequence that relates to engagement with an application.

Usage-centric Evaluation

Hayes et al. [10] propose an online evaluation framework for recommender systems that tackles issues related to the dynamically changing conditions in the wild. Instead of measuring absolute engine performance, it performs a comparative measure of how one recommendation strategy performs against another as part of a real world online system used by a community of users. The authors argue that the relative comparison provides both recommender engines with the same resources

[3]`http://appazaar.net`

and that they suffer from the same negative impact of data changes, thus providing fair evaluation in an online scenario. Swearing and Sinha [23] argue that the effectiveness of recommender engines is not solely determined by the quality of the algorithm. Instead they suggest that trust, explanations, serendipity, and user-customizable filters are important factors for effective recommendation. Also, McNee et al. [16] argue that accuracy metrics alone cannot be used to judge usefulness of recommendations and therefore propose new user-centric directions for evaluating recommender systems. They define three aspects of the recommendation process that accuracy metrics cannot measure: similarity of recommendation lists, serendipity, and importance of user needs and expectations. Pu et al. [17] examine combined criteria for usability and satisfaction and conceptualize a unifying recommender systems evaluation framework, called *ResQue*. It consists of thirty-two questions and fifteen constructs aiming at measuring qualities of recommended items, systems' usability, usefulness, interface and interaction qualities, users' satisfaction with the systems, and influence of these qualities on users' behavioral intentions. In the same line of thinking, Knijnenburg et al. [14] present a framework for user-centric evaluation of recommender systems linking system aspects to user behavior aspects to explain why and how a recommender's user experience evolves. Finally, Konstan and Riedl [15] argue that we need a more diverse set of measures to evaluate the user experience of recommender systems.

In our work, we present a usage-centric perspective and its capabilities for the evaluation of recommender engines that suggest mobile apps.

CONCEPT OF APPFUNNEL

Stages of App Engagement

To ascertain an app's usefulness for a user, we adopted the concept of conversion funnels [2]. The final conversion represents the conversion of a recommended application to one that is used in the long-term. In order to reach this conversion, users go through several stages that represent different events resulting from their interactions with recommended apps. As depicted in Figure 1, we can distinguish four different stages in a user's action sequence for determining a conversion funnel for a mobile application after it was recommended.

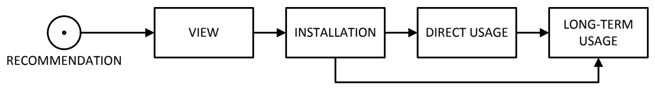


Figure 1. The *AppFunnels*'s stages along a user's app interaction sequence.

After the recommendation of a list of applications to the user, the first stage that an app can reach is the *view stage*. It happens when a user clicks on a particular app from the list of recommended apps. This stage reflects the lowest level of interest that a user might have in an recommended application; it can be elicited by the name, the icon, other users' ratings, or the category of the application. By retrieving application details, he can view additional information like a description, screenshots, or other users' comments.

The *installation stage* appears when the user downloads and installs an application. This stage reflects an even stronger interest in the application. Furthermore, an installation of an app signals that the user actually finds that application interesting enough to try it based on its description and other users' feedback. For some apps, this also means that the user is willing to pay for the application.

Directly after installation, an app might reach the *direct usage stage*. This reveals that a user launches the application shortly after it has been installed, which reflects the next interest level, i.e. he is interested enough to directly try it. Another reason might be that he needs the recommended app in his current context and expects it to be useful. As such, this event is optional and might not appear for applications that are not launched right away. Instead, the user might leave the application and try it later.

Finally, an application might reach the final stage of *long-term usage*. This stage represents the final conversion which reflects the strongest possible level of interest in a recommended app: it reveals that a recommended app turned out to be useful and engaging. In comparison to the previous stages, this one can only be determined from post-processing historical app-usage data, and not from observing a single event resulting from a user's app interaction. As we show in the next section, an app's relative usage time can be used to determine a threshold for considering app usage to be long-term usage.

The four stages in this funnel can only be reached sequentially following the arrows shown in Figure 1. Every stage can be traversed several times, always starting from the *view stage*, if a user has uninstalled an application and retrieves a recommendation again.

Determining Long-term Usage

For determining the long-term usage of an app, we calculate its relative usage $r = u/t \in [0, 1]$, with u denoting the number of days during which the application was used and t denoting the total number of days that the application was installed. The granularity of days is reasonable, since the quotient r for relative usage should address long-term usage and the question is whether a user keeps using an app at all. Aggregating based on days additionally compensates for the effect that app usage varies during the course of a day [6].

The relative usage is influenced by an app's type, meaning that some applications naturally have a higher and others a lower relative usage; e.g. communication apps are used more often than applications providing functionality that is solely useful in particular situations like shopping on the weekend [6]. Based on the data provided through *appazaar* [6], this can be illustrated with the two

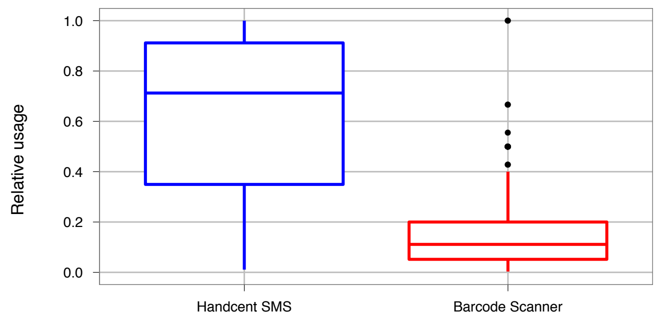


Figure 2. Relative usage of two apps (paired data of 197 users).

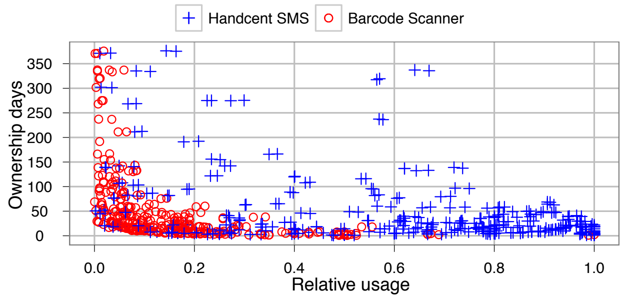


Figure 3. Relative usage versus number of ownership days of two apps (paired data of 197 users).

applications *Handcent SMS* (a popular free messaging application for smartphones) and *Barcode Scanner* (provides users with the ability to scan barcodes containing various types of information such as website links or contact information). Figure 2 depicts the relative usage of these two applications across 197 users. The lower and higher borders of the boxes depict the lower and higher quartile, respectively; we considered only paired cases. The plot shows a significant difference (*Wilcoxon signed-rank test, W=16510, Z=11.77, p<.001, r = 0.59*) in the relative usage of these two apps, with a high relative usage for *Handcent SMS* and a low relative usage for *Barcode Scanner*. The median relative usage for *Handcent SMS* is about 71%, denoting that people use this app nearly 5 days per week; median relative usage for *Barcode Scanner* is 11%, meaning that people use it on less than one day per week.

A user removing an application does not necessarily mean that it has not provided any benefit to the user, especially if the user owned the application for a long period of time. Therefore, such cases require additional analysis of the amount of time that the application was owned by the user and further comparison with other users. For example, if a user uninstalls an application after a period of time that is longer than the median ownership time of all other users, then his relative usage reflects a potential long-term engagement despite the fact that he uninstalled the app. In addition, this approach can also detect events when a user initially does not like an application and uninstalls it after a short ownership time. For example, if an application is installed, used and uninstalled on the same day, it would have a high relative usage of 1.0. For our example, such relative usages are shown in the bottom right corner of Figure 3 for people who used *Handcent SMS* for only a short time. If one particular application is kept by the majority of users for a long time, it can be assumed that the users who have installed the application for only a short time have just tried the application and directly rejected it, thus in this case the high value of the relative usage does not reflect a genuine long-term engagement and should be ignored.

CASE STUDY OF APPFUNNEL

We conducted a case study to examine the validity of our approach and explore possible conclusions that can be drawn based on data collected from the *AppFunnel* framework. Thus, we implemented the proposed framework within a deployed app recommender system and put it to the test in the wild.

Testbed

We used the *appazaar* recommender system provided by Böhmer et al. [5] as our testbed. The system has already been installed by 6,680 users[4]. We do not know any demographic information about our users, but since our testbed *appazaar* is deployed on the *Google Play Market*, we may assume that those people who have installed the *appazaar* application are a good representation of those people who can be expected to use a mobile app recommender system.

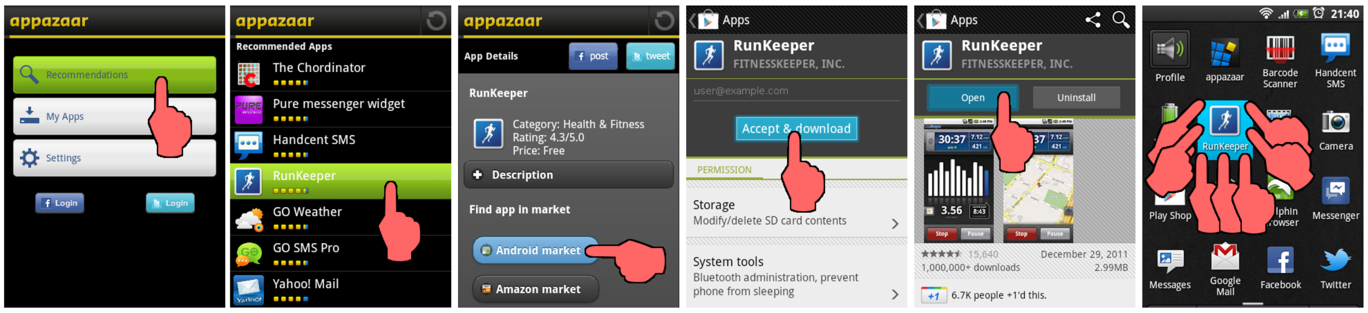
In our testbed we had to add a *view market stage* between the *view stage* and the *installation stage*. This is specific to *appazaar* since it builds upon, but is not integrated into, *Google Play Market* itself. The only way to recommend apps for installation on Android devices is to forward the user to the *Google Play Market* (where he has to grant the app's permission request). Figure 4 sketches the screens a user traverses from the view stage to long-term usage. The *view market stage* is reached when a user decides to review details about an app. The user reaches a screen where he can read other users' feedback about the app and install it. Since the user has to perform an additional click this event reflects a stronger "want-to-try level" of interest towards the application.

Applying Metrics to the AppFunnel

We extended the testbed to collect data about the given recommendations and the corresponding action sequences taken by users as determined by our conversion funnel. As such, for all given app recommendations, we kept track of which stages the recommended apps reached (view / view market / installation / direct usage / long-term usage).

In analogy to well-known and widely-used metrics like click-through-rates or pay-per-click we can define metrics for recommended applications reaching the different stages and conversions between those stages. In our case study we have chosen metrics for the different stages in the form of counters for how many apps have reached

[4] According to Google Play's Android Developer Console.



(a) recommendation (b) view (c) view market (d) installation (e) direct usage (f) long-term usage

Figure 4. Examples of screens a user sees when traversing the stages of our testbed's conversion funnel. This usage-centric action sequence can be captured for evaluation of the different recommender engines.

a certain stage, e.g. the long-term usage metric is the number of apps that have been used in the long-term. Conversion rates are the quotients of two counters of stage events, e.g. the ratio of the number of people who viewed a recommended app to the number of people who used it in the long term.

Recommender Engines under Test

The focus of this paper is on the *AppFunnel* as an evaluation framework and not on investigating new engines themselves. Therefore we have used the recommender engines that already have been developed and deployed within *appazaar* for testing our framework. As shown in Table 1, the engines under test can be classified according to the two dimensions of personalization and context awareness. The personalized engines take information about specific users into account, i.e. their app usage histories. The context-aware recommender engines generate recommendations based on users' current context, e.g. time, location, and previously used apps.

	Non-personalized	*Personalized*
Context-less	– App popularity	– Usage-based CF
Context-aware	– App-aware filtering	– Location-aware CF – Time-aware CF

Table 1. Design space of tested recommender engines.

Within *appazaar*, users' requests for app recommendations have been scheduled to the different recommender engines randomly. This allows for counter-balanced within-subject testing of the engines. All engines that are utilizing collaborative filtering are implemented based on the *Apache Mahout* framework.[5]

It is worth mentioning that not every recommender engine was able to produce results for every request. The system suffers from the cold-start problem [19], in particular for the context-aware engines. When no recommendation could be presented to the user respectively, no user interaction with the recommendation list was possible, and as a result no *AppFunnel*-related data was collected. However, to present apps to the user regardless, the fallback strategy for all engines is to return a random set of applications. Since the interaction with such random data does not contribute to the evaluation of the engines themselves, we do not take into account this data for the presented evaluation framework.

App Popularity

The first recommender engine implemented in *appazaar* is based on apps' popularities measured by their usage. Since installations alone do not reveal much about the quality of an application[6], this engine ranks applications according to their popularity based on their global usage in terms of total number of launches. This engine is neither personalized nor context-aware. Therefore, this engine was able to return results for all issued requests for recommendations, since it did not require any specific additional data.

Usage-based collaborative filtering

The second engine implements collaborative filtering. It utilizes users' app usage data as implicit feedback encoded as binary values for user/item pairs: 1 meaning a user has used an app and 0 meaning a user has never used the app. This engine is personalized, but not context-aware. The core of this engine was implemented using a user-based collaborative filtering algorithm. Therefore, it suffers from the cold-start problem and cannot recommend any applications to users that are new to the system, or to those who did not upload app usage data.[7]

App-aware filtering

This recommender engine is based on the idea that people's usage sessions between screen-on and screen-off have a specific contextual scope. For instance, there is an increased likelihood that people stay with games once they have used a game, or with social apps once they have used a social app [6]. Also, for app usage prediction the preceding app is a strong predictor [22]. Therefore, this app-aware engine recommends applications that are similar to those that have been used recently within the same session. The app-aware filtering is a context-aware but non-personalized approach. Since this engine is based on the apps recently used by the user, this engine has failed whenever the user asked for recommendations without having used an app other than *appazaar* previously in the same session.

[5] `http://mahout.apache.org/`

[6] Jakob Nielsen, `http://goo.gl/KR09G`

[7] Users could opt out from uploading data for privacy reasons.

Time-aware collaborative filtering

The type of apps people use changes during the course of the day, e.g., news applications are more likely to be launched in the morning, and multimedia apps are more likely to be launched in the evening [6]. To exploit this phenomenon, we implemented a recommender engine that incorporates the hour of the day as an additional piece of context information. We extended the previously described collaborative filtering engine following a context-splitting approach [3] and multiplex a user's identity by the 24 hours of the day. As a result, the user-at-time/item pairs denote whether a user has used an application at a specific time of the day or not. This engine suffers from the cold-start problem similarly to the other two engines based on the collaborative filtering algorithm. Its range of possible contextual values is, however, limited to a set of discrete values denoting the hour of the day.

Location-aware collaborative filtering

Location has a high impact on mobile information needs [12], and app usage patterns [24]. Therefore — analogously to the time-aware engine — we also tested an engine that exploits the location of mobile device usage. This engine is also based on the context-splitting approach, modeling pairs of user-at-location/item denoting whether a user has used a specific app at a certain location or not. This engine is personalized and context-aware. This recommender engine suffers from the same problem as the *usage-based collaborative filtering*, because it also uses the collaborative filtering algorithm. Its success rate is further decreased by its location awareness, since there are simply too many different locations defined by geographic coordinates. Locations have been discretized into cells of approx. 30m x 30m (to match e.g. sizes of peoples' homes, offices, or stores). As a result, only a limited number of popular locations include enough usage data that can be used by the collaborative filtering algorithm to produce a successful recommendation; otherwise it just fails.

For the location-aware engine, the data is split to a higher degree than for the time-aware engine. Having only 24 hours in a day allows more application usage data to be used by the collaborative filtering algorithm for a particular hour than for a particular location.

Results

The main purpose of our case study was to examine the capability of the *AppFunnel* framework to evaluate different recommender engines by applying metrics. The following analysis does not aim to find the best performing recommender engine, but to illustrate various observations that can be made by investigating the experimental data that can be collected by means of the *AppFunnel*-based framework.

The data was collected from December 5, 2011, to March 5, 2012, i.e., over a period of three months. In total it contains 287 *AppFunnel* events (204 views, 50 market views, 20 installations, 8 direct usages, 5 long-term usages) contributed by 45 users. The events are distributed across the tested recommender engines as follows: 137 for *app-popularity filtering*, 47 for *usage-based collaborative filtering*, 35 for *app-aware filtering*, 48 for *time-aware collaborative filtering*, and 20 for *location-aware collaborative filtering*.

Within our testbed, the determination of direct usage is performed directly on the users' smartphones. If a user launches a recommended application within five minutes after its installation, this usage is considered to be a direct usage. In order to determine long-term usage of a recommended application, we perform several steps offline. Firstly, we calculate all applications' relative usage across all users who have used it. For each particular application we then pick the lower quartile of users' relative usage values as a threshold. If a particular user's relative usage is higher than this threshold, we consider his usage of this particular app as a long-term usage. Thereby we filter out those app users who had the app installed but did not use it long enough for the application to be considered as engaging long-term usage. Secondly, we calculate the median ownership time for the application across all users who have used it. If a particular user's ownership time is less than the median ownership time, this ownership is filtered out from long-term usage. Thereby we exclude those cases where people have a high relative usage but only had the app installed for a very short time, as discussed earlier.

For a comparison of the recommender engines' performances, the data collected from the *AppFunnel* cannot be applied directly, because of the varying number of successful recommendation lists that each engine has produced. Therefore, we normalized the number of counted *AppFunnel* stages by the number of successful recommendation lists returned per engine. The resulting data represents the average number of events related to recommended applications per successful recommendation list. Since every resulting recommendation list can contain more than one application which people might interact with (see Figure 4, left), the relative number of action sequences per stage can be more than one. This normalization counteracts the fact that not every recommender engine was able to produce a result set for every request. The data is shown in Figure 5.

Conversion Stages

The conversion stage metric depicts how many applications have reached a certain stage in the *AppFunnel*. It appears that the *app-popularity based engine* resulted in the most views of recommended applications. Having a relative number bigger than 1 means that on average people have viewed more than one app from a list of apps recommended by this engine. As for all tested recommender engines, there is a steep drop-off in numbers for the following stages, as Figure 5 shows.

The *usage-based collaborative filtering engine* resulted in the second most views of recommended applications.

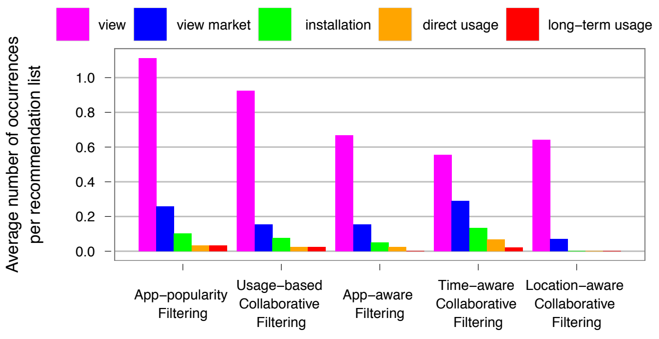


Figure 5. Average number of events counted per recommender engine and *AppFunnel*'s stages.

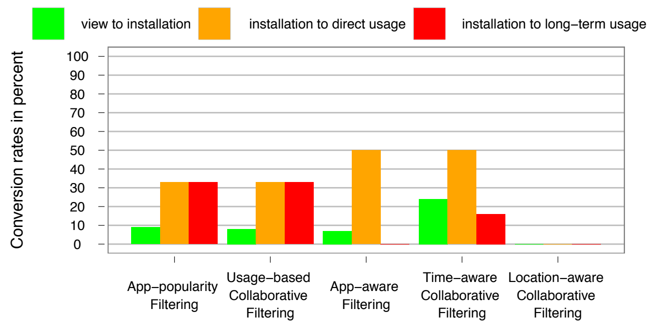


Figure 6. Conversion rates of engines under test. The location-aware CF did not result in any of these conversions (see Figure 5).

However, this engine also shows a steep drop-off after the view event. This engine successfully generated a great initial impression with applications that look interesting enough for the users initially, but are discarded after they are reviewed in detail.

In contrast to the previous two engines, the *app-aware engine* has a lower number of views of recommended apps. In addition, this engine has produced only direct usage and no long-term usage interests, which can also be a result of users just installing similar applications driven by curiosity. Users tried the installed apps immediately, but did not use them regularly in the long term.

The *time-aware collaborative filtering engine* has resulted in the least view events. However, out of all engines, this one has the highest number of action sequences that have reached the three stages of view market, installation and direct usage. Although this engine has recommended applications that resulted in long-term usage, these are fewer than the number of apps that were directly used after they were installed.

The *location-aware collaborative filtering engine* not only has the lowest success rate of all the engines, but it is also the only engine that has not resulted in any installed or used applications. This is interesting, since it even has a slightly higher number of view counts than the time-aware engine. Although the number of views was not less than for the other context-aware engines, there is already a larger gap between the view and the market stage. This engine had no installations, and therefore also had no direct or long-term usage.

Conversion Rates

The conversion rate metric denotes the relative number of action sequences that users follow from one stage to the next. As it can already be seen in Figure 5, the higher the stage within an action sequence, the fewer events occur. Therefore, for all transitions a conversion rate naturally is between 0 and 1 (including endpoints). Although it is possible to calculate conversion rates between all successive stages of a conversion funnel, the most interesting ones related to mobile application usage are: *view to installation*, *installation to direct usage*, and *installation to long-term usage*. These conversion rates are shown in Figure 6. They are based on the absolute and normalized numbers that also have been taken into account for Figure 5.

From the view stage to installations, the engine based on *app-popularity* has a conversion rate of 9%. Overall, for all engines only a few people installed applications that they viewed. In general, the conversion rates from installations to usages — either direct or in the long term — are higher. For the app-popularity engine, both conversion rates (*installation to direct usage* and *installation to long-term usage*) are higher than 30%.

The *usage-based collaborative filtering* engine has a conversion rate of 8% from views to installations. The conversion rates from *installation to direct usage* and *installation to long-term usage* are both higher than 30% — same as the app-popularity based engine. This shows that for these two engines, many users who found apps interesting enough to install them actually also tried them immediately and kept them in the long term.

The *app-aware filtering* engine has a 7% *view to installation* conversion rate which is almost equal to the *app-popularity* and *usage-based collaborative filtering* engines. This engine has not resulted in any *long-term usage* stages; therefore its *installation to long-term usage* conversion rate is 0%. In contrast, the *installation to direct usage* is 50%. It can be inferred that applications recommended by this engine provide a specific functionality, which is useful at the current moment but is not needed on regular basis.

The *time-aware usage collaborative filtering* engine has the highest *view to installation* conversion rate: 24%. Further, its *installation to long-term usage* conversion rate is 16%. Similarly to the *app-aware filtering* engine, it has a high *installation to direct usage* conversion rate of 50%. Since this engine recommends apps that are typically used at the same time of the day in which the recommendation request is sent, it can be expected to have a high *installation to direct usage* conversion rate.

Since there have been no installations for apps that were recommended by the *location-aware recommender engine* (see Figure 5), there are no conversion rates for this engine.

At the bottom line it can be seen that the two contextless recommendation engines resulted in more views of recommended applications than the others. However, the preliminary results of our case study suggest that the two context-aware engines resulted in the highest conversion rates from installation to direct usage, while the two context-less engines resulted in highest conversion from installation to long-term usage.

DISCUSSION

Recommender Engines under Test

It is not surprising that the recommender engine based on apps' popularities resulted in the most views. It is known that naive non-personalized approaches can compete with well-elaborated algorithms [7]. While naive approaches are much easier to design and implement, the more elaborate recommender engines used in this paper require more testing and fine-tuning.

The high number of views of the app-aware filtering engine might be a result of their similarity with some applications that the users have just used in the same session. This engine recommends similar applications, which very often contain similar keywords in their names or have similar icon symbols. Thus, the users might be driven by curiosity to find out more about these new apps.

The bad performance of the location-aware engine might be a result of the same factor that causes the low success rate of this engine: locations defined by geographical coordinates might result in too fragmented usage data that is too sparse and therefore not suitable for a valuable collaborative filtering, which follows from the splitting approach. Optimization of this particular engine is the subject to future work. However, the *AppFunnel* framework allowed us to trace down this issue in the first place. If we had taken a conventional evaluation approach, e.g. solely based on click-through-rates, the location-aware engine would have been ascertained to have a better performance than the time-aware engine. Only the usage-centric *AppFunnel*-based approach enabled us to investigate this issue by keeping track of user actions following on the view of a suggested app. And only this approach enabled us to find that while the location-aware engine resulted in clicks, it did not result in user value in terms of direct or long-term usage.

Since the time-aware engine recommends such applications as are typically used at the same time of day as the recommendations are requested, it was expected to have a high relative number of direct usage.

Two additional events that can be tracked along a mobile app's lifecycle are updates and uninstalls of an application [6]. However, the update does not tell much about a user's engagement with the application, since frequency of updates is specific for each particular app and its developer, and updates can be done automatically without any user intervention. Additionally, we do not consider the removal of an application as feedback. This can also be exploited, but requires further refinements and post-processing, since the removal cannot be taken as an additional stage in the *AppFunnel* as such. Firstly, the removal is more of a negative sort of feedback, and the *AppFunnel* currently is based on positively growing engagement from stage to stage. Secondly, not every user necessarily removes an application he does not need anymore (e.g. he can also remove only the icon of the app from his home screen). Contrary to this, the *AppFunnel*'s stages compulsorily follow from the user's intent.

Application of AppFunnel

It was interesting to see that — irrespective of the location-aware engine — the two context-aware engines caused different user actions than the two context-less engines. The results of our case study suggest that the performance of the different paradigms of recommender approaches are more diverse in terms of stages reached after a recommended item was clicked on, i.e. *installations* and *direct usages* or *long-term usages* are more diverse than *views*, especially in terms or conversion rates. This suggests that the usage-centric evaluation approach — as introduced by *AppFunnel* — enables a more elaborate evaluation of recommender engines that suggest mobile applications and their design goals, e.g. fostering download and installation counts, or providing instant support in the form of apps that can be used directly, or pursuing user benefit in terms of long-term application engagement.

Choosing from AppFunnel's Metrics

Recommender systems are commonly implemented into e-commerce websites for the sake of increasing turnover and revenue. We argue in the line of Konstan and Riedl [15] that incorporating user experience into algorithms will affect the user's benefit from a recommender, and that a more comprehensive set of measures than currently used are required. For the recommendation of mobile applications, the *AppFunnel* goes beyond measuring the quality of a recommender engine in terms of installations. In the case of paid apps, an increase in revenue is already realized as soon as the customer installs a recommended app. This, however, is only an intermediate stage that an app can reach in the *AppFunnel*. Especially when considering that a large portion of installed apps are lying idle [22], it is reasonable to conclude that recommending for installations alone is myopic and should not be the top-priority goal.

Based on *AppFunnel*, we can reason on the quality of a recommender engine beyond commonly applied metrics, like *click-through-rate* and *pay-per-click events*. However, we suggest considering the metrics we introduced as additional metrics for evaluation of recommender engines rather than as a replacement. Our metrics aim to extend evaluation from a vendor's view to a usage-centric view of recommender system performance — i.e., which app was of actual use in the current context or in the long term.

As we have seen in the case study, one metric might conflict with others. One engine might not perform best in all metrics. How to choose and ascertain the metrics has to be decided according to a recommender engine's design goal. As seen in our case study, a context-aware engine might achieve good results when observing direct usage, but have bad results when taking into account long-term usage. This is sufficient for a context-aware recommender engine, since its goal is to address actual user needs, while a recommender engine generally modeling user taste should aim for good performance according to long-term usage.

We did not take into account applications prices. The price might impact whether an app will be installed or not, but this rather is an attribute of the app instead of being inherent to the app usage. Applying the AppFunnel approach one might find a recommender engine recommending expensive apps to a user who only installs free apps with bad view-to-installations conversion rate.

Our case study shows that by applying the *AppFunnel* and incorporating the metrics we introduced, one can investigate the performance of recommender engines for different goals. In our case, we realized that an engine with average performance in terms of clicks performed poorly in terms of usage.

CONCLUSION

In this paper, we presented *AppFunnel*: a usage-centric evaluation-framework for the evaluation of recommender systems that suggest mobile applications. The main part of this framework is the usage-centric concept of a conversion funnel for tracking mobile application engagement: from viewing a recommended application, to installing it, to over using it directly, to using it in the long term. The *AppFunnel* allows for evaluating recommender engines beyond click-through-rates and download statistics. We have implemented and tested our framework based on a system deployed in the wild, and presented results of applying metrics to the *AppFunnel*. Though the testbed's recommender engines are simplistic, we were able to find that the ones addressing the current context of the user show a higher performance in direct usage, while non-contextualized recommender engines result in a higher long-term usage of installed applications.

REFERENCES

1. Z. Ahmet and K. V. V. Mattila. Mobile service distribution from the end-user perspective: the survey study on recommendation practices. In *Proc. of CHI EA '12*, 2012.
2. A. Bagherjeiran, A. O. Hatch, and A. Ratnaparkhi. Ranking for the conversion funnel. In *Proc. SIGIR 2010*.
3. L. Baltrunas and F. Ricci. Context-based splitting of item ratings in collaborative filtering. In *Proc. of RecSys '09*, pages 245–248, 2009.
4. H. Becker, A. Broder, E. Gabrilovich, V. Josifovski, and B. Pang. What happens after an ad click?: quantifying the impact of landing pages in web advertising. In *Proc. of CIKM '09*, pages 57–66, 2009.
5. M. Böhmer, G. Bauer, and A. Krüger. Exploring the Design Space of Recommender Systems that Suggest Mobile Apps. In *Proc. Workshop CARS*, 2010.
6. M. Böhmer, B. Hecht, J. Schöning, A. Krüger, and G. Bauer. Falling asleep with angry birds, facebook and kindle - a large scale study on mobile application usage. In *Proc. MobileHCI*, 2011.
7. P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proc. of RecSys '10*, 2010.
8. C. Davidsson and S. Moritz. Utilizing implicit feedback and context to recommend mobile applications from first use. In *Proc. of CaRR '11*, pages 19–22, 2011.
9. A. Girardello and F. Michahelles. Appaware: which mobile applications are hot? In *Proc. MobileHCI*, 2010.
10. C. Hayes, P. Massa, P. Cunningham, P. Avesani, and P. Cunningham. An on-line evaluation framework for recommender systems. In *Workshop on Personalization and Recommendation in E-Commerce*, 2002.
11. D. Jannach and K. Hegelich. A case study on the effectiveness of recommendations in the mobile internet. In *Proc. of RecSys '09*, pages 205–208, 2009.
12. E. Kaasinen. User needs for location-aware mobile services. *Pers. and Ubi. Comp.*, 7(1):70–79, 2003.
13. A. Karatzoglou, L. Baltrunas, K. Church, and M. Böhmer. Climbing the app wall: Enabling mobile app discovery through context-aware recommendations. *Proc. of CIKM'12*, 2012.
14. B. P. Knijnenburg, M. C. Willemsen, Z. Gantner, H. Soncu, and C. Newell. Explaining the user experience of recommender systems. *User Modeling and User-Adapted Interaction*, pages 1–64, Mar. 2012.
15. J. A. Konstan and J. Riedl. Recommender systems: from algorithms to user experience. *User Modeling and User-Adapted Interaction*, 22(1):101–123, Apr. 2012.
16. S. M. McNee, J. Riedl, and J. A. Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *Proc. of CHI EA 2006*.
17. P. Pu, L. Chen, and R. Hu. A user-centric evaluation framework for recommender systems. In *Proc. RecSys 2011*.
18. R. Rosales, H. Cheng, and E. Manavoglu. Post-click conversion modeling and analysis for non-guaranteed delivery display advertising. In *Proc. of WSDM 2012*.
19. A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proc. of SIGIR 2002*.
20. K. Shi and K. Ali. GetJar mobile application recommendations with very sparse datasets. In *Proc. of KDD '12*, KDD '12, 2012.
21. Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, A. Hanjalic, and N. Oliver. TFMAP: optimizing MAP for top-n context-aware recommendation. In *Proc. of SIGIR '12*, 2012.
22. C. Shin, J.-H. Hong, and A. K. Dey. Understanding and prediction of mobile application usage for smart phones. In *Proc. UbiComp*, 2012.
23. K. Swearingen and R. Sinha. Beyond algorithms: An HCI perspective on recommender systems. *ACM SIGIR 2001 Workshop on Recommender Systems*, 2001.
24. H. Verkasalo. Contextual patterns in mobile service usage. *Pers. and Ubiq. Computing*, 13(5), 2009.
25. W. Woerndl, C. Schueller, and R. Wojtech. A hybrid recommender system for context-aware recommendations of mobile applications. In *Proc. of ICDEW '07*, 2007.
26. B. Yan and G. Chen. AppJoy: personalized mobile application discovery. In *Proc. of MobiSys '11*, 2011.