Interoperability Ranking for Mobile Applications

Dragomir Yankov
Microsoft
Sunnyvale, CA 94089
dragoy@microsoft.com

Pavel Berkhin
Microsoft
Sunnyvale, CA 94089
pavelbe@microsoft.com

Rajen Subba
Microsoft
Sunnyvale, CA 94089
rasubba@microsoft.com

ABSTRACT

At present, most major app marketplaces perform ranking and recommendation based on search relevance features or marketplace "popularity" statistics. For instance, they check similarity between app descriptions and user search queries, or rank-order the apps according to statistics such as number of downloads, user ratings etc. Rankings derived from such signals, important as they are, are insufficient to capture the dynamics of the apps ecosystem. Consider for example the questions: In a particular user context, is app A more likely to be launched than app B? Or does app C provide complementary functionality to app D? Answering these questions requires identifying and analyzing the dependencies between apps in the apps ecosystem. Ranking mechanisms that reflect such interdependences are thus necessary.

In this paper we introduce the notion of *interoperability ranking* for mobile applications. Intuitively, apps with high rank are such apps which are inferred to be somehow important to other apps in the ecosystem. We demonstrate how interoperability ranking can help answer the above questions and also provide the basis for solving several problems which are rapidly attracting the attention of both researchers and the industry, such as building personalized real-time app recommender systems or intelligent mobile agents. We describe a set of methods for computing interoperability ranks and analyze their performance on real data from the Windows Phone app marketplace.

Categories and Subject Descriptors

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

General Terms

Experimentation

Keywords

mobile, apps, ranking, recommender systems

SIGIR'13, July 28–August 1, 2013, Dublin, Ireland.

1. INTRODUCTION

The growing trend of users consuming information through their mobile as opposed to desktop devices requires rethinking some of our approaches to ranking, information retrieval and recommendation. Identifying relevant content on the web is by now a relatively well understood process. In mobile, however, content is primarily consumed through a special and very diverse proxy - mobile apps.

The question of how to provide better utility to users now shifts from discovering and ranking relevant information to discovering and ranking relevant apps that can in turn provide relevant information. This comes with a number of challenges. Firstly, app marketplaces have very limited knowledge of what users ultimately get from an app which they have installed. Apps often aggregate and display content in custom formats which are not transparent to the underlying platform so that proper inference could be conducted. A news app, for instance, may be launched only for the entertainment news which it aggregates or only for the financial news. Secondly, the well established notion of "hubs and authorities" from the web world [4] is also not explicitly present in the case of mobile application.

Here we demonstrate that a simple idea of how we think about apps can alleviate many problems. The idea is that **we should not treat apps independently**, but rather recognize that **there are implicit and explicit dependencies** between them. Identifying these dependencies can help us approach the above challenges. Indeed, if we find out that many users who launch the news app from the example subsequently launch stock trading apps, then it is reasonable to assume that they have consumed financial news through the news app and are now looking for some complementary functionality. We can say that the news app is acting as a hub to multiple financial apps and also that recommending entertainment apps to its user-base will be less relevant than recommending financial apps.

With the above in mind, we introduce the concept of *interoperability ranking*. The interoperability rank of app A quantifies the importance it has to other apps in the ecosystem. We propose several methods for computing the rank and compare their performance on data from the Windows Phone marketplace (Section 4). The presented methods utilize a graph structure derived from user-app interaction logs which reflects the existence of potential connections between apps on the marketplace. We call this graph structure *interoperability graph* (Section 3). First, however, we point out a number of important areas where interoperability ranking can provide valuable input.

2. AREAS OF APPLICABILITY

Marketplace recommendation. In the previous sections we gave examples for this application of interoperability ranking.

Personalized real-time recommendation. With many users having tens of apps, navigating and finding the one to launch can be time consuming [3]. We can build instead a special app - a personalized recommender, which in real time infers and organizes the apps that are most likely to be launched next by a user. Inference can take into account the user context, e.g. time of day, location, apps being launched or other context [3]. Among these features, we find that what apps were launched is very predictive of what is to be launched next, which suggests inherent dependencies between the apps which users utilize. Interoperability ranking can help identify these dependencies.

Intelligent agents. More generally, we can have an *intelligent agent* system which predicts not only the next app to launch but a whole sequence of apps which are relevant and should be launched in a given context. This is similar to the "zero-query mobile IR" systems discussed in [5] which predict what mobile users might need next without such need to be explicitly expressed. Another such system with growing popularity is Google Now. Through a set of rules it decides when to launch one or more special apps (*cards*), e.g. driving to the airport can activate the "traffic card" then the "flight status" card. Again, interoperability ranking can help the agent identify and trigger the right apps for a task.

3. INTEROPERABILITY GRAPH

Dependencies between web pages are explicitly expressed through web links which allows for intuitively building the *link graph*. The graph is then utilized in different ranking algorithms such as PageRank [2] and HITS [4]. Identifying dependencies between apps, however, is harder as there are usually no explicitly declared links. There are emerging efforts that try to catalog apps which provide means for other apps to "deep launch" them [1]. Yet, the number of apps in these catalogs is very limited and one needs to resolve to proxies from which to implicitly infer such links.

A proxy that we explore here are user-app interaction logs and in particular launch patterns of the form (A_i, A_j) - app A_j has been launched immediately after app A_i within a session. By session here we denote the time span between a user turning on their device until they turn it off. Listing (1) shows session logs for two users.

$$\begin{array}{lcl} user_1 & : & session_{11}(A_1, A_2, A_3) \\ & & session_{12}(A_1, A_2, A_3) \\ & & session_{13}(A_1, A_3) \\ user_2 & : & session_{21}(A_1, A_2, A_1) \\ & & session_{22}(A_4) \end{array} \quad (1)$$

The logs show $user_1$ having three sessions in which the launch patterns (A_1, A_2) and (A_2, A_3) appear twice ($session_{11}$ and $session_{12}$), and (A_1, A_3) which appears once ($session_{13}$). From $user_2$ we obtain (A_1, A_2) and (A_2, A_1). Aggregating across the users results in the graphs from Figure 1.

In the interoperability graph each app A_i is represented by a node and each pattern (A_i, A_j) with a directed edge pointing to app A_j. The edge carries the implicit notion of dependence - the user may have launched A_j to complement functionally what they obtained from A_i. In the first graph

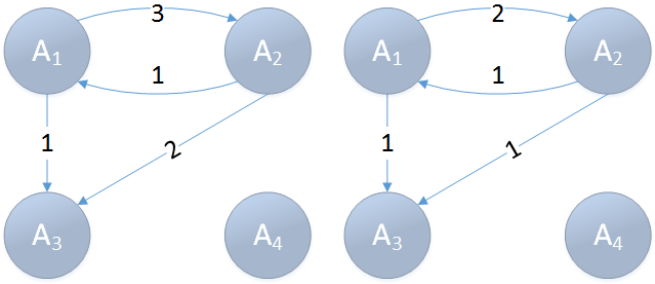

Figure 1: Interoperability graphs for the sessions from Listing (1). Frequency based - each pattern is counted, user based - a pattern is counted only once per user.

(frequency based) we count all patterns when computing the weights of the edges. Often though a user would launch A_j after A_i randomly or while habitually checking the same sequence of apps. We therefore construct the second graph (user based) where each pattern is counted only once per user. In the evaluation we work with this type of graphs.

We exemplify the idea with Figure 2. It shows the interoperability graph for sessions observed over a week. Patterns (A_i, A_j) which appear less than a predefined threshold have been removed. Zooming in the dense area of the graph reveals many patterns. Some are clearly resulting from the vast adoption of certain apps (e.g. social clients) which makes seeing them sequentially in a session also common. Other patterns, however, are more interesting and revealing. We show several among many which we could identify[1]. Thicker edges indicates higher weights for the patterns.

- Figure2, B: *SubwaySchedule*, *BusSchedule*. Both apps are dealing with transportation schedules in a major metropolitan city. Apparently, many people are using both methods of transportation and need to complement their functionality to achieve their intents, e.g. planning commute to work.

- Figure2, C: *Pregnancy1-4*, *ZodiacApp*. The first pregnancy app is one of the popular apps in its category yet there are multiple complaints that it does not track well certain pregnancy aspects. Users are thus complementing it with some of the other pregnancy apps. The data covers a period close between two star signs which probably explains while some of the pregnancy app users tend to launch after that the *ZodiacApp*.

- Figure2, D: *SocialClient1-2*, *PhotoImage1-2*. This example is interesting in several aspects. First, while most existing marketplace recommender systems suggest apps withing the same marketplace category, here we see that there are well defined cross category dependencies, e.g. social apps are related to photo and image processing apps. Second, we find out that *PhotoImage2*, though with lower overall ratings and less installs is launched several times more frequently than *PhotoImage1* by people who also launch *SocialClient1*, which shows that user engagement with an app is often related to its interdependence with other apps, which is not captured by the existing ranking approaches.

[1]The real name of the apps have been obfuscated and replaced with names suggestive of their functionality.

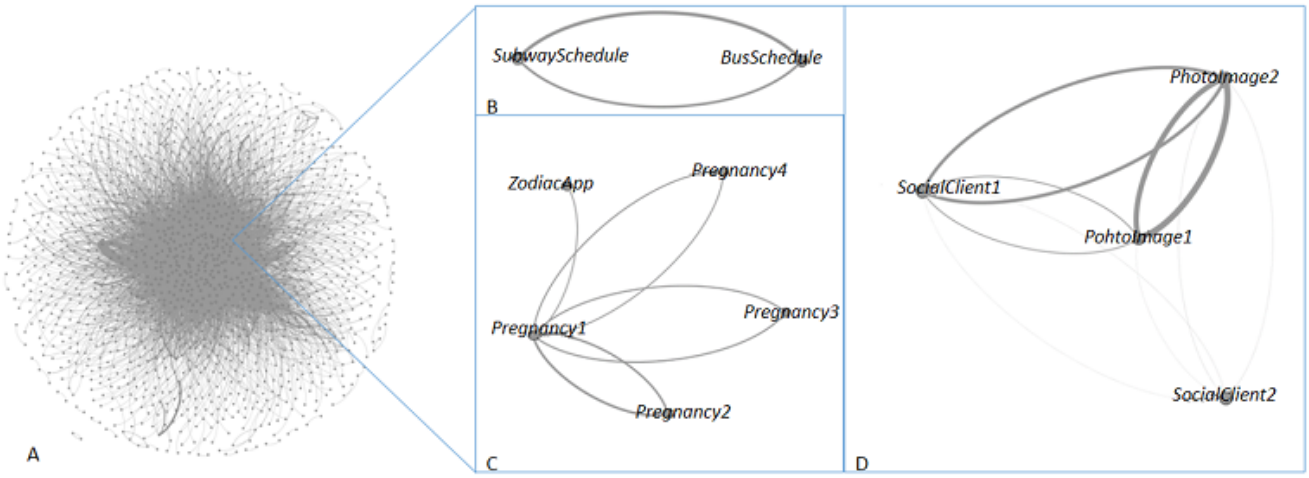

Figure 2: **A**: Interoperability graph built with app sessions observed throughout a week. **B-D**: Zoomed in patterns emerging from the graph - certain pairs of apps are launched disproportionately large number of times (c.f. text for details)

4. METHODS AND EVALUATION

Data and evaluation setup.

The evaluation is done on **four weeks of user-app interactions data** from the Windows Phone app marketplace. We train on week1 and test on week2, then we shift the window, train on week2 and test on week3 and so on. We have removed apps and app patterns which appear less than a predefined number of times in training. After thresholding the data still covers tens of thousands of apps from the marketplace. The comparison here is done primarily with the personalized recommender systems from Section 2 in mind. A user is recommended $k = [1, 10]$ apps (x-axis on Figure 3 and 4). We compute the click-through-rate (CTR) among the top-k recommendations, e.g. if we recommend three apps ($k = 3$) in what percentage of the cases the user actually launched one of them. We report the mean and standard deviation of the results across the three test weeks.

User agnostic methods.

This set of methods compute statistics and **recommend apps from the entire marketplace**. We assume that we do not know what apps each particular user has installed. These methods are good for suggesting new apps to users which others have found useful in a similar context. The performance is summarized in Figure 3.

Static Frequency Based (SFB). In this method we simply compute the top-k most frequently launched apps during the training week and we always predict these apps. I.e. we compute the probability $p(A_j)$ of launching A_j and rank-order the apps based on that probability. This is similar to the "top free/paid/popular" etc. recommendations that every marketplace offers. In predicting that the user will click A_j the method does not take into account what app was launched prior to that. Figure 3 shows that if we recommend ten apps we can achieve 9.8% accuracy, or in other words every one in ten launched apps is among the top ten most frequently used apps on the marketplace.

Static AppRank (SAR). Similar to above the method computes a static rank for all apps on the marketplace and every time predicts the top-k apps with highest rank. The rank is computed similarly to PageRank [2] with a few differences which we found suitable for the purpose of interoperability ranking. In particular, the rank is computed over the interoperability graph which has well defined edge weights while PageRank collapses multiple links from one page to another into a single link.

If M is the transition matrix of the interoperability graph, where m_{ji} is the weight for a pattern (A_i, A_j), then the rank is given by the solution of $R = (1 - d)P + d\hat{M}R$, where $\hat{M}$ is the column stochastic matrix derived from M by normalizing the columns to sum to one and replacing zero columns with the probability vector P, which defines how the residual flow gets distributed among all apps. In the classic PageRank, P is uniformly randomly distributed over all nodes. Here we notice that people often start their sessions with some preferred apps, e.g. social clients or games. To avoid overemphasizing such apps we assign a hundred times lower probability in P to apps which appear too frequently (above some threshold) in the start of sessions. The method improves slightly the SFB method (2.3% lift at $k = 10$, Figure 3).

Conditioned Frequency Based (CFB). Here if a user has launched A_i we check the graph and predict A_j for which the edge (A_i, A_j) has the highest weight among all edges starting from A_i. The method performs a maximum likelihood estimate of $p(A_j|A_i)$ - the conditional probability of launching A_j given that A_i has been launched prior to that.

The performance of the method (Figure 3) and its large superiority over the non-conditional SFB and SAR methods clearly demonstrates the ideas emphasized through the text - apps should not be treated independently as there is valuable predictive signal embedded in the interdependencies.

Conditioned AppRank (CAR). The method takes the top-k predictions of the CFB method and reorders them according to their AppRank, computed as explained for the SAR method. As we can see from Figure 3 the static global rank does not improve on top of the maximum likelihood es-

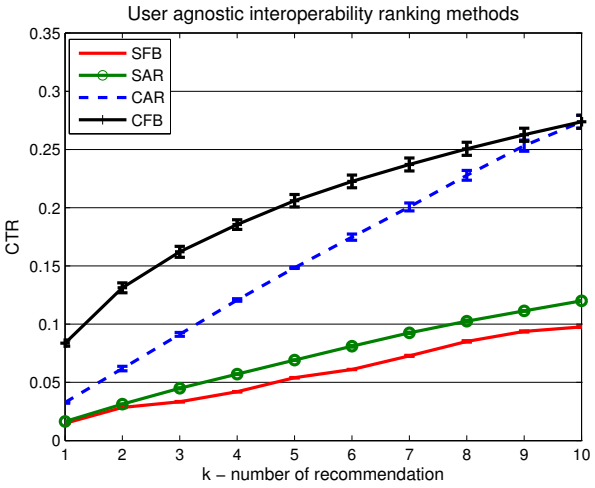

Figure 3: User agnostic predictors.

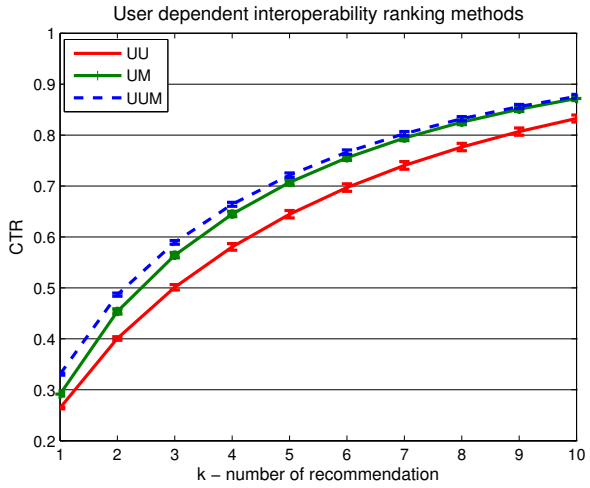

Figure 4: User aware predictors.

timate of CFB. For $k = 10$ both CFB and CAR are identical because CAR simply reorders the results of CFB. Therefore a click on any of the top ten apps recommended by CFB will also in the case of CAR too.

User aware methods.

In this set of methods we are aware of **the apps that a user has installed** and the top-k **recommendations are selected among them**. While these methods produce much higher ctr (Figure 4) and seem very suitable for the personalized recommender system discussed in Section 2 they would not select new apps which others have discovered to be useful in similar context. Hence they are not suitable for diversifying the portfolio of already installed apps which might be important while aiming at better user engagement.

User-User (UU) For each app A_i that a user launches the method predicts the top-k most likely to be launched next apps based on training set frequencies for that same user. If we do not have an estimate for A_i (the app was not launched during the training period) then we predict the top-k apps which the user has launched most frequently on train data. Ties are resolved by random selection. Every week there is a number of new users joining the marketplace. For these users we have no frequency estimates from the training data. For them predictions are made by choosing uniformly at random among their installed apps.

The method does not utilize the interoperability graph or any information from the marketplace. It is fully personalized. One can correctly argue that instead of interdependence patterns it detects patterns where users habitually check the same apps. Its results are significantly better than the user agnostic methods - 83.3% ctr among the top ten recommendations compared to 27.4% for the CFB method.

User-Marketplace (UM). Similar to the CFB method UM uses the interoperability graph to select the top-k A_j which maximize $p(A_j|A_i)$ from marketplace perspective with the additional constraint that all selected A_j should be among the installed apps by the user. The improvement compared to the results for the UU method (ctr 87.2% vs 83.3% for $k = 10$, Figure 4) comes to demonstrate again that app interoperability patterns do exist on the marketplace and can be very useful when there is no user history to compute user specific launch probabilities from.

User-UserMarketplace (UUM). The method is a combination of the UU and UM methods. For a particular user who has launched A_i, if we have an estimate of the top-k apps A_j which maximize $p(A_j|A_i)$ then we select them as prediction. Otherwise, we resolve to estimates from the marketplace similar to UM. The method as seen on Figure 4 improves between $1\% - 4\%$ over the UM method, where the improvement is especially notable for small values of k.

The results remain stable across the three test weeks with the static user agnostic methods naturally showing less variance than the conditional and the user aware methods.

5. CONCLUSION

We studied the concept of interoperability rank - a measure of the importance of apps to other apps on the marketplace. A number of methods were presented for computing the rank. Their performance was demonstrated on a large real world data set of user-app interaction logs. The results reveal that app interdependencies can provide valuable predictive signal in identifying apps with complementary functionality which is in turn essential when aiming for better user utility and engagement.

6. REFERENCES

[1] `http://handleopenurl.com/`.

[2] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the seventh international conference on World Wide Web*, WWW7, pages 107–117, 1998.

[3] P. Coppola, V. Della Mea, L. Di Gaspero, D. Menegon, D. Mischis, S. Mizzaro, I. Scagnetto, and L. Vassena. The context-aware browser. *IEEE Intelligent Systems*, 25(1):38–47, Jan. 2010.

[4] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, Sept. 1999.

[5] T. Sakai. Towards zero-click mobile ir evaluation: knowing what and knowing when. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '12, pages 1157–1158, New York, NY, USA, 2012. ACM.