Predicting The Next App That You Are Going To Use

Conference Paper · February 2015

DOI: 10.1145/2684822.2685302

CITATIONS
99

READS
4,518

4 authors, including:

Ricardo Baeza-Yates
University Pompeu Fabra
650 PUBLICATIONS **32,944** CITATIONS

SEE PROFILE

Di Jiang
Baidu Online Network Technology
29 PUBLICATIONS **303** CITATIONS

SEE PROFILE

Predicting The Next App That You Are Going To Use

Ricardo Baeza-Yates
Yahoo Labs
Barcelona, Spain
rbaeza@acm.org

Di Jiang*
CS and Eng. Dept.
HKUST, Hong Kong, China
dijiang@cse.ust.hk

Fabrizio Silvestri
Yahoo Labs
Barcelona, Spain
silvestr@yahoo-inc.com

Beverly Harrison
Yahoo Labs
Sunnyvale, US
beverlyh@yahoo-inc.com

ABSTRACT

Given the large number of installed apps and the limited screen size of mobile devices, it is often tedious for users to search for the app they want to use. Although some mobile OSs provide categorization schemes that enhance the visibility of useful apps among those installed, the emerging category of homescreen apps aims to take one step further by automatically organizing the installed apps in a more intelligent and personalized way. In this paper, we study how to improve homescreen apps' usage experience through a prediction mechanism that allows to show to users which app she is going to use in the immediate future. The prediction technique is based on a set of features representing the real-time spatiotemporal contexts sensed by the homescreen app. We model the prediction of the next app as a classification problem and propose an effective personalized method to solve it that takes full advantage of human-engineered features and automatically derived features. Furthermore, we study how to solve the two naturally associated cold-start problems: app cold-start and user cold-start. We conduct large-scale experiments on log data obtained from Yahoo Aviate, showing that our approach can accurately predict the next app that a person is going to use.

Categories and Subject Descriptors

J.0 [**Computer Applications**]: GENERAL.

General Terms

Design, Experimentation, Performance.

Keywords

Aviate, Prediction, Mobile App, Machine Learning.

*This work was done while the author was an intern at Yahoo Labs, Barcelona, Spain

WSDM'15, February 2–6, 2015, Shanghai, China.

http://dx.doi.org/10.1145/2684822.2685302.

1. INTRODUCTION

Mobile devices (smartphones and tables) are ubiquitous in today's lives. This high popularity also corresponds to a huge growth in the availability and usage of mobile applications (commonly referred to as apps). Mobile applications are very easy to install and this usually correspond to having mobile phones with a very large number of apps installed. An empirical study conducted by Yahoo Aviate team shows that on average there are 96 apps installed on each mobile device. This large number of apps installed calls for the design of new paradigms aimed to manage the installed apps. In particular, one of the major issues associated with the high number of apps installed on a smartphone is that of their visibility. When a user needs a particular app it is not always available immediately and the search through the large number of installed apps might take lots of time. This is the reason why homescreen apps are becoming more and more popular. Homescreen apps act as an intelligent layer between the underlying mobile operating system and the user interface. They manage the installed apps in a highly personalized manner rather than simply relying on manual (or trivial) categorization schemes.

The basic building block that we consider at the heart of an effective personalized management of apps is a personalized app prediction mechanism. In other words, the most important feature of our homescreen app is that of anticipating the user needs even before the user would click on the relative icon on his/her mobile phone. Ideally, once the predictor has made its guess the homescreen app should show the application already opened as the user picks up the phone and unlocks it. Even if this scenario might sound futuristic, we show here that to predict which app a user is going to open, is actually feasible. Our solution leverages the sequence of real-time spatiotemporal contexts that are continuously sensed by the homescreen app (in this case, the Yahoo Aviate application).

App Prediction Problem is formally defined as follows:

Given a list of installed apps $\{a_{u1}, a_{u2}, ..., a_{un}\}$ *by a user* u *on his/her phone and the user's spatiotemporal context* C*, the problem of app usage prediction is to find an app* a_{ui} *that has the largest probability of being used under* C*. Specifically, we aim to solve the problem:*

$$\max_{a_{ui}} P(a_{ui}|C, u), \forall i, 1 \leq i \leq n;$$

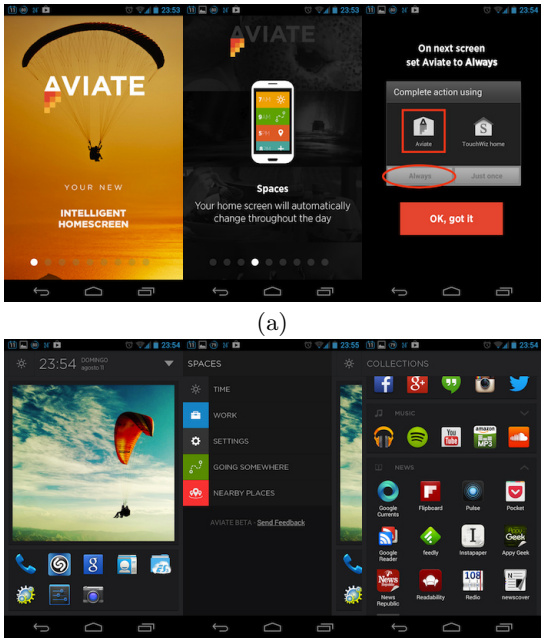

Figure 1: Screenshot of Homescreen App Aviate.

We model the problem of app usage prediction as a supervised classification problem. Through a comprehensive study of the log data obtained from Aviate (see Figure 1), we propose several features from two different perspectives: basic features (i.e., signals) obtained from the mobile phone's sensors and the session features capturing sequential patterns of app usage. Based on these features, we design highly scalable algorithms to train the prediction models.

In addition, the application prediction functionality in Aviate (or in a homescreen app in general) suffers from two inherent cold-start problems: app cold-start and item cold-start. The first one deals with the problem of not knowing anything about past usages of that app by the mobile phone user. The latter deals with the problem of generating predictions when the user first install Aviate. In this paper we also propose several techniques to alleviate the effects of cold start. Conducting extensive experiments on Aviate log data, our experimental results show that the user's app usage is predictable based on the user's usage history and the wisdom of the crowd.

The contributions of this paper are summarized as follows:

- We conduct a large scale comprehensive study of mobile app usage.
- We propose several domain-specific features for app usage prediction. We studied how to conduct efficient app usage prediction and we conduct a preliminary study on how to handle the app cold-start and user cold-start problems.
- We conduct extensive and large scale experiments based on usage information recorded into a log from Aviate. We show that our method outperforms existing ones.

The remainder of this paper is organized as follows. In Section 2, we review the related work. In Section 3, we describe the empirical study of app usage. In Section 4, we describe the methods for app prediction. In Section 5, we discuss how to handle the cold start problem. Finally, we present experimental evaluations in Section 6 and conclude the paper in Section 7.

2. RELATED WORK

The problem of predicting app usage has been already proposed by other researchers in the past. In particular, Tan *et al.* [16] proposed a prediction algorithm for predicting mobile application usage patterns. They conducted experiments on the Nokia MDC dataset which contains a small group of 38 users and their experiments show promising results. Huang *et al.* [6] studied the problem of pre-loading the right apps in memory for faster execution or can pop the desired app up to the mobile's home screen. Similarly to our work, Huang *et al.* [6] exploit contextual information such as time, location, and the user profile, to predict the user's app usage using the Nokia MDC dataset already mentioned. Yan *et al.* [18] designed a app preloading method by using contexts such as user location and temporal access patterns to predict app launches with 34 volunteers. Pan *et al.* [13] proposed a different prediction problem: finding the most likely mobile application that a user will install. They exploited social information coming from friends of the user in a social networks. Parate *et al.* [14] designed an app prediction algorithm that predicts which app will be used next without sensor context. Zou *et al.* [20] proposed some light-weighted Bayesian methods to predict the next app based on the app usage history. Krishnaswamy *et al.* [8] developed a general-purpose service that runs on the phone and discovers frequent co-occurrence patterns indicating which context events frequently occur together. Liao *et al.* [9] designed a widget that is able to predict users' app usage by constructing temporal profiles which identify the relation between apps and their usage times.

Our work is also related to app analytics and app recommendation. Kai *et al.* [17] proposed a framework called App Developer Inspector, which aims to effectively profile app developers in aspects of their expertise and reputation in developing apps. Jiang *et al.* [7] presented a framework to provide independent search results for Android apps with semantic awareness. Yin *et al.* [19] proposed an Actual-Tempting model that captures such factors in the decision process of mobile app adoption. Henze *et al.* [5] reported on five experiments that were conducted by publishing apps in the Android Market. Based on these outcomes, the authors identified factors that account for the success of experiments using mobile application stores. Lin *et al.* [11] presented a framework that incorporates features from version descriptions into app recommendation. Lin *et al.* [10] described a method that accounts for nascent information culled from Twitter to provide relevant recommendation in such cold-start situations.

Our work differs from the previously proposed methods in several ways:

1. To the best of our knowledge, this work is the largest scale of evaluation for app usage. In existing work, the scale of user study is usually limited to less than 40 users, while our study is four orders of magnitude larger;

Table 1: Examples of Aviate Log Data (the user ID is anonymized to ensure privacy).

User ID	App Action
xx8ae648c10	{"ts":"2014-01-10 12:27:39","et":"App_Opened"} ["com.skype.raider"]
xx8ae648c10	{"ts":"2014-01-10 12:36:09","et":"App_Opened"} ["com.android.dialer"]
xx8ae648c10	{"ts":"2014-01-10 12:47:41","et":"Location_Update"} ["37.393093","-122.079788","20.000000","0.000000"]
xx8ae648c10	{"ts":"2014-01-10 12:57:42","et":"Context_Triggered"} ["Work"]

2. We achieve the highest precision for app prediction. The precision of the state-of-the-art is usually far below 90%;

3. Besides achieving high precision, our method is highly scalable for big data;

4. We study the cold start problems that are still open in app usage prediction and we study both perspectives of cold start: users and apps.

3. APP USAGE CHARACTERIZATION

As this is the first time that such a large scale analysis is performed, we present here some basic statistics of the Aviate sample log data. The log sample contains more than 60 million records with about 200K anonymized users that have a total of more than 70K unique apps by January 2014. An example of the events recorded in the log are shown in Table 1. For example, the first log entry indicates that "Skype" (the app) *com.skype.raider* was opened by user *xx8ae648c10* at *2014-01-10 12:47:39*. The third entry shows that the user's location is moved to the GPS location *(37.393093, -122.079788)*. The fourth entry tells us that Aviate's working mode is switched to *Work*.

Aviate records 10 types of actions in total. The distribution of those actions is depicted in Figure 2. As expected, *App Open* represents a large fraction of app actions. This actually confirms that the app prediction problem could impact significantly on Aviate by improving its user experience. Furthermore, we observe that the fraction of *App Installed* event is relatively low but is non-negligible (around 250K times in our dataset). Just to put this into perspective every ten times a user changes location he or she installs an app. For this reason the app cold-start problem is an important one that needs a particular attention.

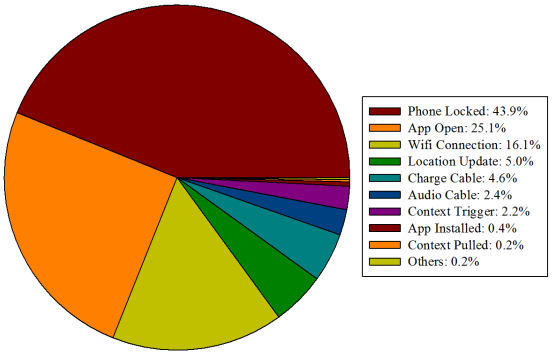

Figure 2: Distribution of Actions.

4. APP USAGE PREDICTION

In this section we discuss the problem of predicting the next app that a given user is going to open as she will pick up her mobile phone. We start by presenting a naive prediction strategy based on apps popularity in 4.1. We discuss our set of features, namely *basic* and *session* features in Sections 4.2 and 4.3, respectively. Then we discuss how to train the prediction model and how to generate predictions in Section 4.4.

4.1 Predictions by Popularity

At first sight the app prediction problem might seem to be straightforwardly solvable by always predicting the most popular app. While it is obvious that a popularity score computed globally among all the users is not going to work in practice, it might seem reasonable to consider a *per-user* popularity score. We discuss two kinds of popularity-based features: *Global Frequency* and *Timeslot Frequency*. *Global Frequency* represents the number of times that an app has been opened by a particular user. *Timeslot Frequency*, instead, is the number of times that an app has been opened within a specific timeslot of the day (e.g. at Noon). We consider timeslots of granularity ranging from 1 day to 1 second and based on the two kinds of frequencies we predict the next app by selecting the one with the highest (global or timeslot-based) popularity. We run a test to evaluate the effectiveness of popularity-based approaches. We extract popularity statistics on 80% of the log data and we test the methods on the remaining 20%. Results show that using *Global Frequency* we can achieve an average precision of up to 16.7%. We observe similar results also when we consider timeslot-based statistics. Results of predictions using various timeslot granularities are shown in Figure 3. Using 1 day based timeslots we reach a precision of 16.3%. Reducing the timeslot to 1 hour the prediction precision decreases to 15.2% and the precision further decreases to 12.6% when the timeslot size is 1 second. Results shown in Figure 3 demonstrate that differently from what the intuition suggests, solely relying on timeslots is not effective to find any useful pattern in real-life app usage scenarios. We further investigate the prediction performance of considering a week as a cycle. We calculate the frequencies of each app at each day in a week and using the most frequent one at the corresponding date. The prediction precision is 10.5% showing that app usage does not have very strong periodic phenomenon in terms of week.

In order to understand why popularity-based prediction might fail, we study the distribution of the *activeness* of the apps, in other words how an app is used. The activeness of an app a is denoted as Δ_a, which is formally defined as follows:

$$\Delta_a = \frac{\sum_{i=1}^{S} \frac{F_{ai}}{F_i}}{S},$$

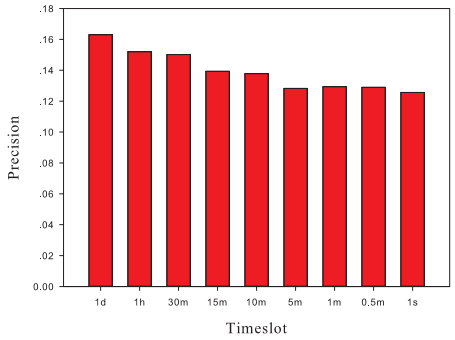

Figure 3: App Usage Prediction by Timeslot Frequency.

where S is the number of timeslots, F_{ai} is the frequency of a in timeslot i and F_i is the frequency of all apps in the i-th timeslot. Δ_a reflects a's activeness across different timeslots in each day. We show app activeness with its activeness rank (in decreasing order of activeness) for the 30 most popular apps in Figure 4. In the plot each line corresponds to a different timeslot granularity. We find that the activeness of the apps on mobile device follows a power-law distribution. A small fraction of dominant apps take a large proportion of the whole activeness. Most of these dominant apps are related to basic services like browser, phone and communication. These dominant apps are frequently used in each timeslot, making the other apps hard to predict if only the frequencies are considered. Hence, we need to look for more effective features to approach the problem.

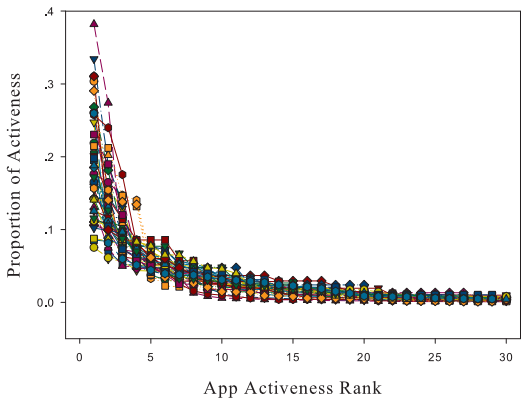

Figure 4: Activeness Proportion v.s. Activeness Rank

4.2 Basic Features

Since app usage is strongly related to the user's spatiotemporal context, we discuss some basic features that can be directly obtained from the sensors of mobile devices. These basic features defining the user's spatiotemporal contexts are the following: *Time*, *Latitude*, *Longitude*, *Speed*, *GPS Accuracy*, *Context Trigger Context Pulled*, *Charge Cable*, *Audio Cable*. The first five features are self-explanatory. *Charge Cable* and *Audio Cable* indicate whether the corresponding cable is plugged into the mobile device. *Context Trigger Context Pulled* indicate whether the corresponding Aviate user has switched the ambient in which he operates (e.g., "work", "home", etc). Among these features, only *context trigger* and *context pulled* are peculiar of Aviate whereas the other features are obtained from the OS. These features help to represent the spatiotemporal context of actions related to the app open event. The feature *Time* is normalized between 00:00:00-23:59:59, in order to capture daily app usage patterns. As a side note, we have also tried to add the App Activeness feature to our set of basic features. The contribution of avtiveness has not been significant therefore we have removed it and we are not going to consider it in the experiments we show next.

4.3 Session Features

Basic features do not capture latent relations between app actions. Intuitively, and as also reported by Huang *et al.* [6], the correlation between sequentially used apps may have a strong contribution to the accuracy of app prediction. In this section we propose a method to extract session features by considering the sequential relationships existing among app actions.

We exploit the ideas presented in *word2vec* [12]. In word2vec documents are modeled as sequences of terms and a context of a term is defined to be the k immediately preceding and following terms in the document. Likewise, we model documents with sequences of app events but we cannot simply choose the preceding and following actions as the context of an app given that they may be associated with events happened a long time back in the past (or ahead in the future). Based on this observation, we propose a Gaussian based method to identify the context of each app action. The context is defined as the set of other actions that are conducted before or after the current one within a time period whose duration is obtained by sampling from an empirically defined Gaussian distribution. For example, in Table 1 we show a context relative to *App Open:com.android.dialer* formed by the following two app actions: *App Open:com.skype.raider*, and *Location Update:(37.393093, -122.079788, 20.000000, 0.000000)*.

Our objective is to find distributed representations of these features that are useful for predicting the surrounding actions of the current context. More formally, given an app action τ and its context C_τ, the objective is to maximize the average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{\tau' \in C_t} \log p(\tau'|\tau_t), \tag{1}$$

where C_t is the context of τ_t and T is the number of app actions. The probability $p(\tau'|\tau_t)$ is calculated as follows:

$$p(\tau'|\tau_t) = \frac{e^{V_{\tau'} \cdot V_{\tau_t}}}{\sum_{t=1}^{T} e^{V_{\tau'} \cdot V_{\tau_t}}} \tag{2}$$

where $V_{\tau'}$ is the distributed representation of τ' and V_{τ_t} is the distributed representation of τ_t.

In order to enhance the efficiency, the full softmax in the above formula is approximated by the hierarchical softmax [12]. In order to efficiently compute the distributed representations, we parallelize the training procedure using a MapReduce paradigm. The Mapper and Reducer are presented in Algorithms 1 and 2. The MapReduce procedure iterates for a predefined number of times and the output vectors are the session features for mobile apps.

Algorithm 1 Mapper of Distributed Representation

```
load the vectors of app actions from the previous iteration;
initialize the Huffman tree based on the loaded vectors;
for each app action a do
    draw a temporal gap g_1 from Gaussian distribution G;
    draw a temporal gap g_2 from Gaussian distribution G;
    consider the actions between g_1 and g_2 as the context C_a;
    train a based on C_a;
end for
for each app action action a do
    emit(key=ID_a,value=Vector_a);
end for
```

Algorithm 2 Reducer of Distributed Representation

```
for each app action a do
    calculate average and normalize the corresponding vectors;
end for
flush updated vectors to HDFS;
```

Based on Algorithms 1, and 2, we define distributed representations of the following six app actions: *Last App Open*, *Last Location Update*, *Last Charge Cable*, *Last Audio Cable*, *Last Context Trigger*, *Last Context Pulled*. The corresponding distributed representations of the six actions are in turns considered as the session features for the app prediction method we use.

4.4 Building Prediction Models

By merging basic and session features for each *App Open* event, we obtain the training instances for the prediction model. In order to achieve a balance between effectiveness and efficiency, we propose the Parallel Tree Augmented Naive Bayesian Network (PTAN) as the prediction model. PTAN is a parallel version of the Tree Augmented Naive Bayes (TAN) [3], which has been proposed in order to remove the strong assumptions of independence in native Bayes and exploit the correlations among attributes. This model captures the latent correlations between different features and can be easily deployed on parallel computing frameworks such as Hadoop. The training procedure of PTAN can be divided into two phases: (1) parallel structure training and (2) parallel parameters estimation. The first phase is to learn the structure of the Bayesian network. The procedure of constructing the PTAN bayesian network from data is as follows:

1. Based on the training data from all users, we compute the conditional mutual information between any two attributes $\mathbf{f_x}$ and $\mathbf{f_y}$ given the app a. Conditional mutual information is defined as follows:
$$I_p(\mathbf{f_x}, \mathbf{f_y}|a) = \sum_{\mathbf{f_x},\mathbf{f_y},a} P(f_x, f_y, a) \log \frac{P(f_x, f_y|a)}{P(f_x|a)P(f_y|a)}.$$
2. Build a complete undirected graph in which the vertices are the features. Annotate the weight of an edge connecting feature $\mathbf{f_x}$ and $\mathbf{f_y}$ by $I_p(\mathbf{f_x}, \mathbf{f_y}|a)$.
3. Build a maximum weighted spanning tree for the complete undirected graph. The maximum spanning tree problem can be transformed to the minimum spanning tree problem simply by negating edge weights [2]. Kruskal's algorithm [4] is used to solve the minimum spanning tree problem.
4. Transform the resulting undirected tree to a directed one by choosing a root variable and setting the direction of all edges to be outward from it. Construct a Bayesian network by adding a vertex labeled by app variable a and adding an arc from a to each feature vertex.

The highest computational cost is associated with the first and the third step in the above algorithm. If we assume there exist m training instances and each instance has n features then the first step has complexity of $O(n^2m)$. The third step has a complexity of $O(n^2 \log n)$. Since $m \gg \log n$, the first step is the bottleneck of building the Bayesian network structure. Hence, in PTAN, we parallelize the first step again using MapReduce. The Mapper for parallelizing the first step is presented in Algorithm 3 and the Reducer can be straightforwardly developed by aggregating the values based on the keys.

Algorithm 3 TAN Structure Learning Mapper

```
for each app action entry {u, a, f_1, f_2, f_3, ..., f_n} do
    for each feature pair (f_i, f_j) do
        emit(key={f_i, f_j, a},value=1);
        emit(key={f_i, a},value=1);
        emit(key={f_j, a},value=1);
    end for
end for
```

After learning the structure of PTAN, we estimate a set of parameters for each individual user, in order to achieve the personalization effect. The conditional probability is estimated as follows:

$$P(f_i = k|pa(f_i) = j) = \frac{N_{ijk} + N'_{ijk}}{N_{ij} + N'_{ij}}, \tag{3}$$

where f_i is a feature, $pa(f_i)$ is the set of the parents of f_i, N_{ij} is the number of times $pa(f_i) = j$, N_{ijk} is the number of times $f_i = k$ given $pa(f_i) = j$ in the training set, and N'_{ij} and N'_{ijk} are the smoothing parameters that we set both to 0.5 by default. Smoothing is fundamental for the prediction performance when some user contains very few training instances for an app. Furthermore, parameters estimation for each user is also parallelized. The Mapper is presented in Algorithm 4 and the Reducer can be straightforwardly developed by aggregating the values based on the keys. When the parameters for the Bayesian network have been estimated we compute the probability that a user u is going to use mobile app a_{ui} when it finds herself within the context $C = \{f_1, f_2, ..., f_n\}$ as follows:

$$\begin{aligned} &P(a_{ui}|f_1, f_2, ..., f_n) \\ &\propto P(a_{ui})P(f_1, f_2, ..., f_n|a_{ui}) \\ &= P(a_{ui}) \prod_{i=1}^{n} P(f_i|pa(f_i)) \end{aligned} \tag{4}$$

Based on this equation we consider the app that has the highest probability as the prediction result. It is straightforward to transform this prediction algorithm into a top-k recommendation one just by sorting apps according to the computed probability scores.

Algorithm 4 TAN Parameter Learning Mapper

1: **for** each action long entry $\{u, c, a_1, a_2, a_3, ..., a_n\}$ **do**
2: **for** each pair $(a_i, pa(a_i))$ **do**
3: emit(key=$a_i, pa(a_i)$,value=1);
4: emit(key=$pa(a_i)$,value=1);
5: **end for**
6: **end for**

5. COLD START PROBLEMS

In this section, we discuss how to handle two of the main challenges we face when homescreen apps that are deployed in real-life environments. The first problem is the *App Cold Start*, which happens when a user installs a new app on her device. The second problem is the *User Cold Start* that arises when a user installs and open the homescreen app for the first time. User cold start is more severe than the first one given that in order to reduce the risk of app abandonment it is important to provide high quality personalized predictions already from the first interactions with the homescreen app. We discuss how to tackle these two cold start problems in Sections 5.1 and 5.2.

5.1 App Cold Start

For a newly installed app a_i, we have no user-specific information available on the app. In particular the probability of opening an app for a given user u, $P(a_{ui})$, is unavailable. On the other hand, $P(f_i|pa(f_i))$, the prior probability for a given feature, can be obtained from information on other users. Therefore, for newly installed apps, how to estimate its $P(a_{ui})$ is critical. We present the app activeness for newly installed apps in Figures 5 (Daily) and 6 (Hourly). We can see that a significant fraction of the newly installed apps is very active within the first few hours. After this period of time the newly installed apps are used significantly less. In contrast, some apps are used frequently and for a long time after their installation. Hence, in order to better estimate $P(a_i)$, we categorize apps as *short-term apps* and *long-term apps* depending on their activeness longevity.

In order to capture the temporal prominence of each app we fit app usage data into a Beta(α, β) variable and to differentiate apps by their temporal significance we use the excess kurtosis [15] to evaluate the temporal peakedness of each app. The excess kurtosis ϱ of a Beta(α, β) distribution is defined as follows:

$$\varrho = \frac{6[\alpha^3 - \alpha^2(a\beta - 1) + \beta^2(\beta + 1) - 2\alpha\beta(\beta + 2)]}{\alpha\beta(\alpha + \beta + 2)(\alpha + \beta + 3)}. \quad (5)$$

An app with a high ϱ value indicates that it is likely to be a short-term app while an app with a low ϱ value is likely to be a long-term app. Through using ϱ, we can categorize the apps into short-term apps and long-term app. Table 2 shows some examples after categorization. As an example, we can observe that while communication apps are usually long-term game apps tend to have a shorter life span.

Short-term App. We assign to short-term apps a $P(a_{ui})$ value of $\overline{P}(a_i)$, which is the average opening frequency obtained from other users' historical information. After a fixed time period (two hours, for example) we can replace $\overline{P}(a_i)$ with the actual app's usage information for that particular user, namely $P(a_{ui})$.

Long-term App. For long-term apps, we exploit the well-known concept of "wisdom of the crowd". When an app has been opened very few times its $P(a_i)$ should approximate the average value coming from other users. As the number of open events grow we can switch to the actual $P(a_{ui})$. Hence, we used Bayesian average [1] to effectively blend actual usage information with that coming from other users' history. The equation for calculating $P(a_{ui})$ is therefore given by:

$$P(a_{ui}) = \frac{C \cdot \Omega_{a_i} + O_{a_{ui}}}{C + O_u}, \quad (6)$$

where $O_{a_{ui}}$ is the number of openings of the app a_i by the current user U, O_u is the sum of all app openings by u, and Ω_{a_i} is the average opening probability of a_i by all other users. C is a constant whose value is set to match the expected variation between data sets containing the history from different users. The value of $P(a_{ui})$ for an app with a small number of open events tends to be closer to the average opening probability of that same app by all other users. The more openings an app has received by the current user, the more accurate the probability estimation is.

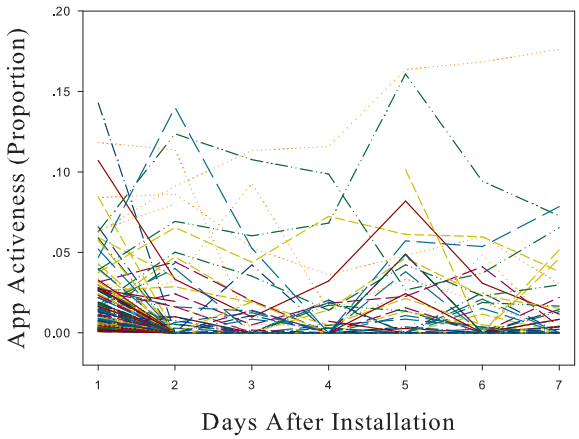

Figure 5: Daily App Activeness (Proportion) After Installation.

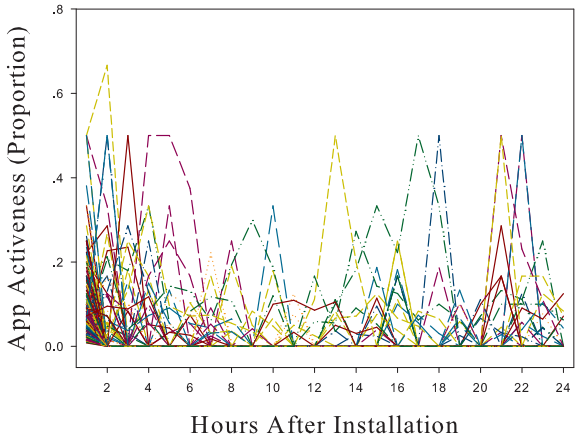

Figure 6: Hourly App Activeness (Proportion) After Installation.

Table 2: Examples of Short-term and Long-term Apps.

Short-term Apps
jp.gree.jackpot
com.rockstargames.gtasa
com.sq.dragonsworld
com.cocoapps.elbomberman
com.zhan_dui.animetaste
com.sweettracker.smartparcel
com.mobie.catholiclife
com.partyplay.xxl
com.king.candycrushsaga
com.battlelancer.seriesguide
Long-term Apps
com.quixey.android
com.google.android.talk
com.whatsapp
mobi.mgeek.TunnyBrowser
com.fitbit.FitbitMobile
com.spotify.mobile.android.ui
com.google.android.apps.authenticator2
com.facebook.katana
com.myfitnesspal.android
com.google.android.gm

On the other hand, when an app has received just a small number of openings, its estimated $P(a_{ui})$ value is close to its unweighted average opening probability computed on all the other users.

5.2 User Cold Start

In this subsection, we discuss two different strategies to tackle the user cold start problem. In Section 5.2.1 we discuss the *Most Similar User Strategy* essentially based on collaborative filtering. In Section 5.2.2 we discuss the *Pseudo User Strategy* that is designed to synthesize an appropriate pseudo-history for the new user so that the PTAN model can be effectively trained on this surrogate usage data.

5.2.1 Most Similar User Strategy

When a new user installs homescreen apps such as Aviate, it is not true that we do not know anything about her. In particular, we know her apps inventory and using this information we can identify the most similar user from the set of known users by using metrics such as, for instance, Jaccard similarity between the new user and the known ones.

After the most similar user has been found, we can use this user's model to predict the behavior of the new user. We name this strategy as the *Most Similar User Strategy*. In fact, the most similar user may have an app inventory that is very different from that of the user under consideration. In some extreme cases the app inventory can be totally different from those installed by known users and the most similar user strategy would not have any sensitive advantage over a purely random strategy. Of course we can resort to put a threshold on the minimum acceptable similarity. While this strategy improves the average accuracy of the method it also limits the coverage, i.e., the number of users for which recommendations can be generated.

5.2.2 Pseudo User Strategy

In order to solve the problem of the strategy presented above we propose the *Pseudo User* strategy, which consists in generating a "pseudo history" used in place of the actual one to train the PTAN model for the new user. Let us assume there are k users $U = \{u_1, u_2, u_3, ..., u_k\}$ and let us further assume that each user u has an app inventory I_u. When a new user with app inventory I_{new} installs the homescreen app, we obtain I'_u for each u by only keeping the apps in I_{new}. We want to find a subset of users U' satisfying the following conditions:

$$\min \sum_{u \in U'} |c(u)|$$
$$\text{subject to} \quad \bigcup_{u \in U'} I'_u = I_{new} \ .$$

The cost of an user $c(u)$ is defined as the inverse of its similarity to the current user, i.e., $c(u) = 1/J(I_{new}, I_u)$. The idea is to find a small amount of similar users whose app inventories cover all the apps in the inventory of the new user. It is straightforward to prove that the problem is NP-hard (it is essentially a weighted set-covering problem) and the $O(\log |I_{new}|)$-approximation algorithm we use to generate the pseudo user is presented in Algorithm 5. After obtaining U' we aggregate log entries for each user in U' and we filter out the *App Open* events for those apps that are not in I_{new}. What it is left is considered to be the *Pseudo User* data used in PTAN training.

Algorithm 5 Building Pseudo User

1: $U' \leftarrow \emptyset$
2: $I \leftarrow \emptyset$
3: **repeat**
4: choose $u \in \mathcal{U}$ minimizing $\frac{c(u)}{|I \cup I_u - I|}$
5: let $U' \leftarrow U' \cup u$
6: let $I \leftarrow I \cup I'_u$
7: **until** $I = I_{new}$

6. EXPERIMENTS

In this section we measure the performance of the proposed method. Experiments are conducted on Aviate log data collected from October 2013 to April 2014. We randomly choose 480 active Aviate users for our experiments. For each user, we use 80 percent of the data as training data and the remaining 20 percent as testing data. For all the methods we extract features (both basic and session) using events within a sliding window of 12 hours, as we assume that any event only influence future events that happen within 12 hours. In Section 6.1 we measure the performance of the propose app prediction algorithm. In Sections 6.2 and 6.3, we evaluate the performance of our solution to the app and user cold start problems, respectively.

6.1 App Usage Prediction

As we have stated in the previous sections, we modeled the app prediction problem as a classification problem and we opted for a parallelized version of the Tree Augmented Naive Bayes (TAN) algorithm to build the classifier. In order to assess the effectiveness of PTAN we compare it with the best ML algorithms that are suitable for this problem, namely Naive Bayes, SVM, C4.5, and Softmax Regression

(i.e., Multinomial Logistic Regression). In the experiments we report the score for precision, that is the percentage of correctly predicted opened apps over the total number of app opened by users in the test data.

Figure 7 shows the performance of different methods in the case of a model trained when the context is made up of only the basic features. In this case, C4.5 achieves the highest precision of 34.9%. We investigated the reasons why C4.5 results is the best method for basic features and we conjecture that is due to that some apps have a very clear fingerprint in terms of sensor usage that a classification tree is able to surface. For example, by analyzing the decision tree, we found that when the charge cable is plugged in, the predicted apps were some video games or using bluetooth. Intuitively, game apps or file transmission apps usually consume more battery energy, therefore when you plug your charge cable (and you do not wait too much time before taking an action), it is very likely you are going to use one of the two classes of applications mentioned above.

We then evaluate the hypothesis we made that sequence-based patterns are useful for building more effective classifiers. For methods such as Naive Bayes, C4.5, PTAN and Softmax Regression, the session features are effective to boost the performance. Therefore, these results confirm the assumption that the sequential correlations between app actions are critical for app usage prediction. Combining basic features and session features is an effective strategy for Bayes methods. With both types of features, Naive Bayes achieves a precision of 76.3% and *PTAN achieves the highest precision of 90.2%*. We also studied the impact of dominant apps (i.e, the most frequent apps) in app prediction. By filtering the top five dominant apps for each user, we reevaluate the performance of the five methods. The performance of Naive Bayes with basic and session features is heavily influenced, giving a precision of 39.3%. Among the five methods, PTAN still achieves the highest precision of 85.7% with all the features. Therefore, we can conclude that while dominant apps are easy to predict using even just a Naive Bayes classifier, when they are removed a strong signal can still be obtained by opportunely combining the other features we have designed (in particular features about event sequences).

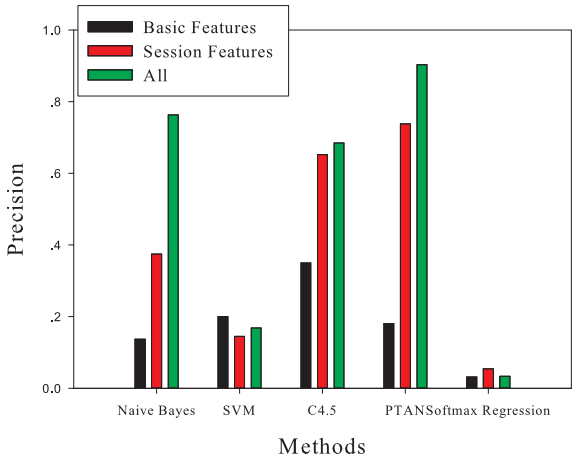

Figure 7: App Usage Prediction (All Apps).

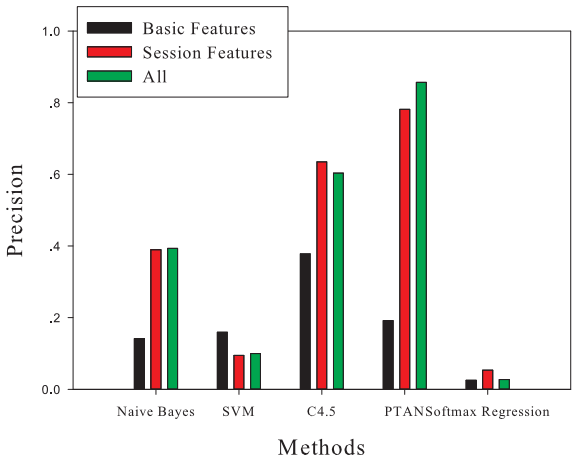

Figure 8: App Usage Prediction (Dominant Apps Filtered).

In order to comprehensively measure the performance of the proposed method, we compare it with the state-of-the-art counterparts. Although some methods described in Section 2 are related to our work, some of them are not comparable with PTAN because they use some additional information. We select EWMA [16], CPD [16], LU-2 [20], LUT [20], and BN [20] as comparison baselines as they can be used on our dataset. We evaluate these baselines by varying the different values of the parameters that characterize those methods. Figure 9 shows the performance of EWMA and CPD on the full set of apps (including dominant ones) and Figure 10 shows the performance of EWMA and CPD when dominant apps have been filtered out from the dataset. With the full set of apps EWMA achieved the highest precision of 16.7%. When dominant apps were filtered out, instead, CPD has a precision of 19.5%, which is the highest in this case. We further observe that EWMA performs better with all apps while CPD performs better when the dominant apps are filtered. Figure 11 shows the performance of LU-2, LUT and BN. LU-2 shows the best performance with all apps, achieving a precision of 15.7% while BN achieves the highest precision of 10.1%.

From the various results shown we can conclude that PTAN significantly outperforms the five baselines we consider by a large margin. We remark that the experiments done using the techniques presented in the papers describing the baselines were done on a much smaller dataset containing usage data on 38 users in total. Our user set is one order of magnitude larger while the dataset is several order of magnitude larger. Nevertheless the performance reported for papers on the baselines methods are consistent with the ones we report here.

6.2 App Cold Start

We now assess the performance of the method discussed in Section 5.1 and we evaluate its effectiveness in alleviating the app cold start problem. For each app installment, we process the original dataset by splitting it into two subsets: one subset containing all the events referred to the period before the app installment and one subset containing the remaining events. We then build a model for predicting app opening events contained in the second file by using events

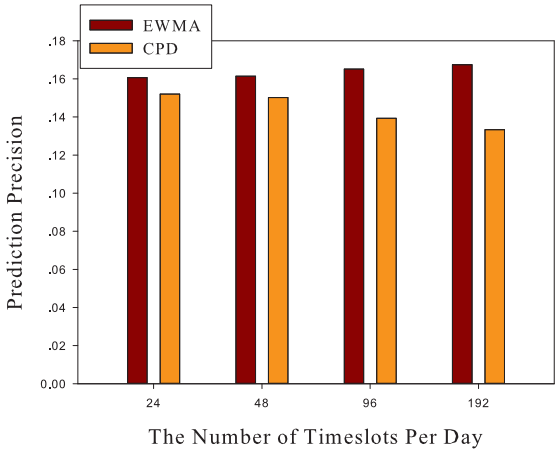

Figure 9: App Usage Prediction of Baselines (All Apps).

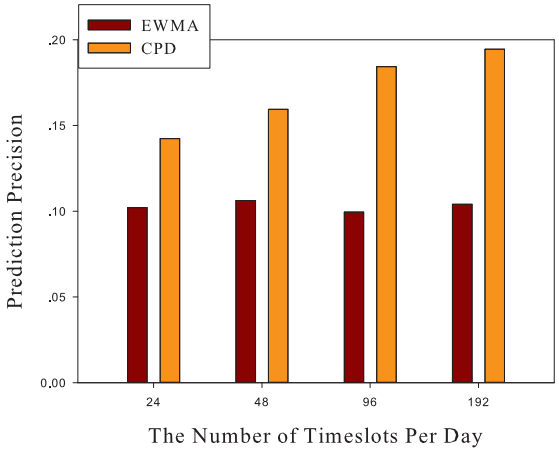

Figure 10: App Usage Prediction of Baselines (Dominant Apps Filtered).

in the first file. For short-term apps (those that are heavily used in the moments immediately following their installation), the precision of app prediction improves from 86.3%[1] to 87.1% after applying the strategy described in Section 5.1. In particular the precision for new apps is about 91.3% and for existing apps is about 86.25%. The prediction quality for existing apps is thus not heavily hurt. The results confirm that in the case of short-term apps we usually observe opening events during the first day after they are installed. The number of opening events steadily decreases afterwards. Therefore, assigning a relatively high probability for these apps is useful for improving the performance. Note that the increase in precision is significant since users open many apps each day and the opening of the newly installed apps may only happen in a relatively small fraction of the cases. Indeed, the percentage of newly installed apps in the test data is only 1.42%.

The precision we measure in the case of long-term apps is 89.3% when we use the general method, but no newly

[1]Precision percentages are different from the previous section as we are reporting results on a different partitioning of the dataset created to test cold start methods.

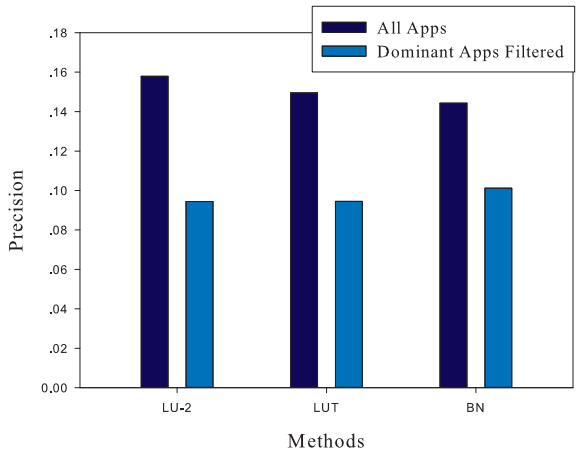

Figure 11: App Usage Prediction of Baselines.

installed app is correctly predicted in this case. By applying the proposed cold-start strategy the precision measured increases up to 90.3% on average. More accurately the precision is 91.1% for new apps and 89.26% for existing apps, also confirming that in this case the precision on existing apps is not heavily hurt.

6.3 User Cold Start

Experimental results show that the *Most Similar User Strategy* achieves an average precision of 32.7%. We present the correlation between app inventory similarity and app prediction precision in Figure 12. We observe that the app inventory is useful yet not very effective. There are two main reasons that may explain this result. First, it is not easy to find a user with highly similar inventory. In our dataset, the Jaccard similarity between two different app inventories is typically low. We found an average value of 0.121465 ($\pm$0.038955) and a median of 0.117647. Second, even when the similarity is high the corresponding precision increase is not always satisfactory. In many cases, we observe that the precision is almost zero even when similarity of the two app inventories is relatively high.

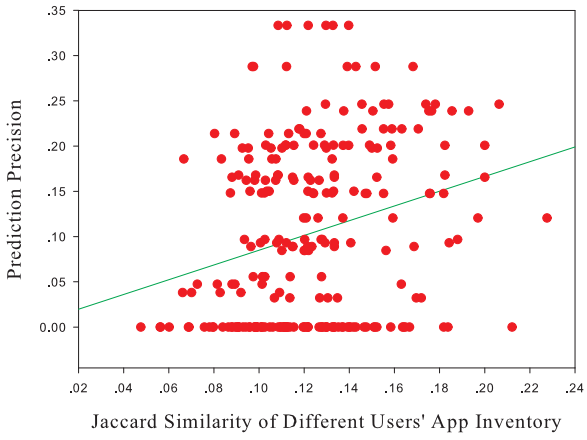

Figure 12: App Prediction By Similar User.

On the other hand, the average precision of the *Pseudo User Strategy* is 45.7%. Therefore we confirm that estimating usage data about apps in a new user inventory using this strategy is effective to boost the performance of the prediction model.

Besides reporting the effectiveness of the two proposed user cold start strategies we also studied after how many days the app prediction model can achieve acceptable prediction precision. The result is presented in Figure 13 where we show that even after just one single day of data about the user has been collected the prediction's precision increases up to more than 56%. Precision further increases up to 81.5% when two days of historical data are considered.

We can conclude that *Most Similar User Strategy* and *Pseudo User Strategy* are both useful but only in the very same day the homescreen app has been installed. After that, for a typical user, high-precision prediction models can be trained using the real (i.e., collected) usage data from that user.

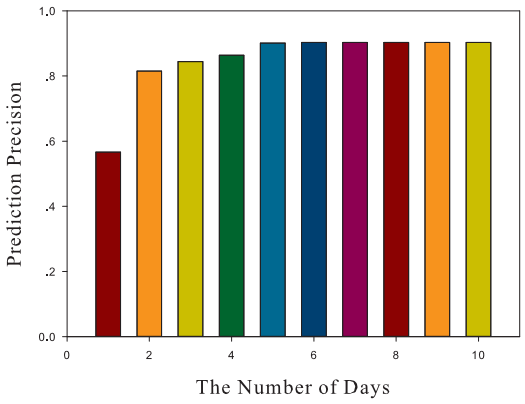

Figure 13: App Prediction Precision v.s. Number of Days.

7. CONCLUSIONS

In this paper we have proposed a new methodology to predict what is the next mobile app that a user is going to open. We conducted a comprehensive analysis on a very large scale mobile log containing events recorded by Aviate, the Yahoo's homescreen app. We proposed a set of basic and session features that are designed to capture the sequential correlation between different actions involving mobile apps; e.g., plugging the charging cable. Based on these features, we built a Parallel TAN model to efficiently solve our app prediction problem. We further studied how to alleviate the app cold start problem and the user cold start problem. We conducted a wide spectrum of experiments to study the parameter sensitivity and effectiveness of the proposed method. Experimental results demonstrated that app usage is predictable even though a naive strategy based on popularity features fail to achieve high effectiveness. The proposed method outperformed the state of the art by a large margin.

In the future we plan to further improve the prediction performance at cold start. In particular the problem to improve the performance of user cold start app prediction remains open. Regarding app cold start it is an open problem that of optimizing the performance of the predictor in the presence of a mix of cold and non-cold apps. Finally, it is related the problem of finding the top-k applications that a user would open instead of the next one. The problem, then, could be cast into a learning to rank rather than into a classification one.

8. REFERENCES

[1] George EP Box and George C Tiao, *Bayesian inference in statistical analysis*, vol. 40, John Wiley & Sons, 2011.

[2] David Eppstein, *Spanning trees and spanners*, Handbook of computational geometry (1999), 425–461.

[3] Nir Friedman, Dan Geiger, and Moises Goldszmidt, *Bayesian network classifiers*, Machine learning **29** (1997), no. 2-3.

[4] John C Gower and GJS Ross, *Minimum spanning trees and single linkage cluster analysis*, Applied statistics (1969), 54–64.

[5] Niels Henze, Martin Pielot, Benjamin Poppinga, Torben Schinke, and Susanne Boll, *My app is an experiment: Experience from user studies in mobile app stores*, International Journal of Mobile Human Computer Interaction (IJMHCI) **3** (2011), no. 4, 71–91.

[6] Ke Huang, Chunhui Zhang, Xiaoxiao Ma, and Guanling Chen, *Predicting mobile application usage using contextual information*, Proceedings of the 2012 ACM Conference on Ubiquitous Computing, ACM, 2012, pp. 1059–1065.

[7] Di Jiang, Jan Vosecky, Kenneth Wai-Ting Leung, and Wilfred Ng, *Panorama: A semantic-aware application search framework*, Proceedings of the 16th International Conference on Extending Database Technology, ACM, 2013, pp. 371–382.

[8] Shonali Krishnaswamy, Joao Gama, and Mohamed Medhat Gaber, *Mobile data stream mining: from algorithms to applications*, Mobile Data Management (MDM), 2012 IEEE 13th International Conference on, IEEE, 2012, pp. 360–363.

[9] Zhung-Xun Liao, Po-Ruey Lei, Tsu-Jou Shen, Shou-Chung Li, and Wen-Chih Peng, *Mining temporal profiles of mobile applications for usage prediction*, Data Mining Workshops (ICDMW), IEEE 12th International Conference on.

[10] Jovian Lin, Kazunari Sugiyama, Min-Yen Kan, and Tat-Seng Chua, *Addressing cold-start in app recommendation: latent user models constructed from twitter followers*, Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval, ACM, 2013, pp. 283–292.

[11] _____, *New and improved: modeling versions to improve app recommendation*, Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval, ACM, 2014, pp. 647–656.

[12] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean, *Distributed representations of words and phrases and their compositionality*, Advances in Neural Information Processing Systems, 2013, pp. 3111–3119.

[13] Wei Pan, Nadav Aharony, and Alex Pentland, *Composite social network for predicting mobile apps installation.*, AAAI, 2011.

[14] Abhinav Parate, Matthias Böhmer, David Chu, Deepak Ganesan, and Benjamin M Marlin, *Practical prediction and prefetch for faster access to applications on mobile phones*, Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing, ACM, 2013, pp. 275–284.

[15] Gamini Premaratne and Anil Bera, *Modeling asymmetry and excess kurtosis in stock return data*, Illinois Research & Reference Working Paper No. 00-123 (2000).

[16] Chang Tan, Qi Liu, Enhong Chen, and Hui Xiong, *Prediction for mobile application usage patterns*, Nokia MDC Workshop, vol. 12, 2012.

[17] Kai Xing, Di Jiang, Wilfred Ng, and Xiaotian Hao, *Adi: Towards a framework of app developer inspection*, Database Systems for Advanced Applications.

[18] Tingxin Yan, David Chu, Deepak Ganesan, Aman Kansal, and Jie Liu, *Fast app launching for mobile devices using predictive user context*, Proceedings of the 10th international conference on Mobile systems, applications, and services, ACM, 2012.

[19] Peifeng Yin, Ping Luo, Wang-Chien Lee, and Min Wang, *App recommendation: a contest between satisfaction and temptation*, Proceedings of the sixth ACM international conference on Web search and data mining.

[20] Xun Zou, Wangsheng Zhang, Shijian Li, and Gang Pan, *Prophet: what app you wish to use next*, Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication, ACM, 2013, pp. 167–170.