Mobile App Recommendation with Sequential App Usage Behavior Tracking

Yongkeun Hwang, Donghyeon Lee, Kyomin Jung

Department of Electrical and Computer Engineering, Seoul National University, Korea
{wangcho2k, donghyeon, kjung}@snu.ac.kr

Abstract

The recent evolution of mobile devices and services have resulted in such plethora of mobile applications (apps) that users have difficulty finding the ones they wish to use in a given moment. We design an app recommendation system which predicts the app to be executed with high accuracy so that users are able to access their next app conveniently and quickly. We introduce the App-Usage Tracking Feature (ATF), a simple but powerful feature for predicting next app launches, which characterizes each app use from the sequence of previously used apps. In addition, our method can be implemented without compromising the user privacy since it is solely trained on the target user's mobile usage data and it can be conveniently implemented in the individual mobile device because of its less computation-intensive behavior. We provide a comprehensive empirical analysis of the performance and characteristics of our proposed method on real-world mobile usage data. We also demonstrate that our system can accurately predict the next app launches and outperforms the baseline methods such as the most frequently used apps (MFU) and the most recently used apps (MRU).

Keywords: Mobile App, Recommendation system, Usage prediction, Feature extraction, Distance learning

1 Introduction

Mobile devices such as smartphones and tablets have become popular to the point of ubiquity in everyday life. For example, mobile devices have become the primary method to access Internet for more than half of global mobile users [1]. Consequently, the use of mobile applications (apps) is also growing rapidly. In the United States, Android and iPhone users spend about 30 hours per a month using apps, and they used 26.8 apps a month on average recently [2].

One approach to mitigate the inconvenience is to predict the next apps that are most likely to be used, and to recommend them to users using some user interfaces such as a widget. Users can easily select what they want on the recommended app list, greatly improving user experience. Furthermore, pre-fetching predicted apps into the memory works as a viable method to improve the user experience as well by reducing delays in app launches. If the prediction can be made accurately, the user can access the desired apps more conveniently and quickly. Thus, the accurate prediction of next apps becomes a critical problem in recommending or pre-launching apps to enhance the user experience.

Naive and intuitive methods for this prediction problem are to recommend the most frequently used apps (MFU) or the most recently used apps (MRU). Generally, the MRU method is implemented in practical mobile operating systems such as Android and iOS. Although those methods give users some convenience, they do not exploit prior app usage patterns or other information such as sensor readings, limiting their prediction accuracies.

There are a number of previous researches on more sophisticated method to recommend the next app. The common idea is to learn patterns of previous app usage data and make a prediction by finding the appropriate apps from the learned patterns that are most similar to the given circumstance. Although most of them have shown higher prediction accuracies than the MRU or MFU method, there are a number of limitations such as the lack of flexibility of the proposed models, a naive feature construction. Some methods need to be trained from all user's data so the collection of user's mobile usage data on a centralized system should be available. However, this can induce user privacy concerns from the collective use of user's mobile usage data [3], thus these methods would be less favorable to the users.

In this paper, we propose a new method of mobile app recommendation that aims to provide more precise app usage prediction and not to induce a concern about providing privacy sensitive usage information to the centralized system. Our recommendation system utilizes a variety of accessible information as features by logging corresponding circumstances when an app is used. In particular, we focus on the sequence of previous app usage patterns, thus we introduce App-

*Corresponding Author: Kyomin Jung; E-mail: kjung@snu.ac.kr
DOI: 10.3966/160792642019052003016

Usage Tracking Feature (ATF), an intuitive and efficient method to represent implicit sequence information. After a feature extraction, those features will be combined and expressed in the multidimensional space to apply a *k*-Nearest Neighbor (*k*-NN) algorithm, and we apply a metric learning on the gathered data before the *k*-NN classification task to ensure that the distance metric is effective among all features. We aim that our method can be implemented on each user's mobile device easily and ensure that our system does not need to collect or be trained from other user's data so that our system can be deployed without user privacy concerns. In experiment, we evaluate the performance of the proposed recommendation system by conducting comprehensive empirical experiments on real world mobile usage data from 121 volunteers. We show that our method outperforms the previous works including the baseline methods such as MFU and MRU. We also present the characteristics of our method on a variety of parameter settings and feature combinations and app categories.

The remaining parts of our paper are organized as follows. In Section 1.1, we briefly review the literatures on mobile app usage prediction and app recommendation. Then we define the problem formally and give an overview of our method in Section 2. Our methodology of feature construction and recommendation algorithm are presented in Section 3 and 4, respectively. In section 5, we show a series of experimental results and analyses on our method. Finally, we draw a conclusion on Section 6.

1.1 Related Work

A number of studies have addressed the problem of predicting the user's app usage [4] as a mean of improving user experience. The list of predicted apps can be recommended to a user by creating their shortcuts on user interfaces such as a launcher or a widget. For example, Zhang et al. [5] and Keshet et al. [6] developed an adaptive launcher which displays predicted next app candidates, and Parate et al. [7] created a widget that consisted of adaptive shortcuts of predicted apps. These adaptive user interfaces allow users to find desired apps easily. In addition, the predicted app can be pre-loaded into the device's memory to reduce the time of app execution. Yan et al. [8] showed that prefetching predicted apps effectively reduced the overall app startup time and Parate et al. [7] also applied the next app prediction to app prefetching, in addition to the adaptive shortcuts. These works are closely related with ours, as our method can also be applied in these scenarios as well.

The mobile context information represents situations or activities of the user. It is crucial for providing users with personalized services on mobile [9-10] and it can also be used to collect characteristics of app usage [11]. Thus, finding the effective mobile context information for the app prediction is a key question in this field. There are two types of the context information; explicit and implicit information [4]. Explicit information such as the time and the geographical location is readily available on most mobile devices, so it has been widely exploited from the early studies [8, 12]. In following works, implicit information which is obtained from user's app usage pattern, has been studied. Combined with explicit one, implicit information enables the model to forecast the user's app usage more accurately. One example of the implicit information is the history of app launches. Bohmer et al. [13], Shin et al. [14], and Xiang et al. [15] found that the last used app was effective on app prediction. Similarly, several researchers including Parate et al. [7], Kim and Mielikäinen [16], and Keshet et al. [6] confirmed that the sequence of recent app launches was also useful. These findings led researchers to model the user's behavior more precisely by using the sequence of app use. For instance, Liao et al. [17] used transitions between apps to construct an implicit feature for modeling app use pattern into a vector space. Baeza-Yates el al. [18] suggested the *app actions* which describes the events such as the change of sensor readings (e.g. location updates recorded by GPS), in addition to the recent app launches. Importance of implicit information shown in these works motivated us to design ATF using history of app usage.

The prediction model plays an important role to accurately predict the user's preference of the next app as well. Several types of prediction model have been suggested in previous works. For example, Kamisaka et al. [19] and Shin et al. [14] adopted a Naïve Bayes model with hand-crafted features from all available mobile log data. Zhang et al. [5] constructed a Bayesian Network with an assumption that each app use depends on time, date, location, and the previous app. Zhu et al. [10] suggested that two major types of user preference models by assuming that each type of context feature is conditionally independent or not. Near neighbor (NN) methods such as *k*-NN are also used in a number of previous works, including [17] and [20]. There have been comparisons on performance between models as well. Kim and Mielikäinen [16] compared the conditional log-linear (CLL) model and the *k*-NN based model on their experiments. Baeza-Yates el al. [18] presented a comparative result between several prediction methods including Tree Augmented Naïve Bayes (TAN) and decision tree based on C4.5 algorithm.

2 Setup and Process Overview

In this section, we present the setup and characteristics of the mobile app recommendation systems, and then describe a brief overview of our recommendation process. The objective of an app recommendation system is to predict a few specific

mobile apps that are most likely to be launched next. The mobile app recommendation system records every instance of app usage and corresponding *context* information at that moment. Those aggregated data are used as the training data in the learning phase, and the system recommends a few apps that are most likely to be launched from the learned patterns.

2.1 Setup

First, we define a terminology, the *context* information, which represents the certain circumstances data in which a mobile app is used. Upon the execution of a mobile app, the app recommendation system gathers all the accessible information that represents the environment at that moment. Explicit information like the time, location, and battery status and implicit information on app usage are such examples of the context information and each item of context information can be represented as a feature. We later introduce how to obtain the features in detail.

We define a d-dimensional vector $x \in \mathbb{R}^d$ as a feature vector that concatenates all context information for a single instance of app usage. For example, if we use time, location, and the headset usage as a binary value as the context information, then the feature vector is expressed as $\mathbf{x} = [\mathbf{x}_{\text{Time}}, \mathbf{x}_{\text{Location}}, \mathbf{x}_{\text{HeadsetOn}}]$, which concatenates each piece of context information. Then the corresponding used app is expressed as its app ID number $y \in \mathbb{N}$, which will serve as a label of the feature vector. Thus, each instance of app usage $(x; y)$ is treated as a data point (feature values) with a label (app ID number) in a d-dimensional space.

The app recommendation system is provided with such app usage data to learn the user's app usage pattern and then recommends a list of $R \in \mathbb{N}$ candidate apps that are the most likely to be used next. If the user launches an app in the recommended list, we say that the recommendation is *hit*, and otherwise *missed*. The goal of the app recommendation system is to increase the hitting probability so that users can easily choose the app that they want to execute at that time.

Now, we clarify the characteristics of our app recommendation system. First, the recommendation only targets installed and used apps since our app recommendation system aims to provide suggestions for users to easily launch the next app which they want to use. This differs from the app recommendations such as [21] and [22], which aims to recommend the new apps to the users who might have never used before. Second, we do not recommend previously unused apps on the user's mobile device since we aim the prediction of the frequently and repeatedly used apps. Third, to preserve user privacy, we design the recommendation system to use either explicit or implicit context information from the device itself only and not to use information from other devices or centralized system. We also note that the user-based recommendation methods including the Collaborative Filtering (CF) which utilizes collective user patterns, are inadequate in our recommendation objective since recommendation systems that collects user's data can pose serious user's privacy concerns about using the system [3]. Particularly, it is known that many mobile users have concerns about collecting their privacy-sensitive data in the mobile apps and sharing the data with service providers [23]. Moreover, some users even try to restrict the app to use their mobile context information such as location [24]. To avoid such misgivings on deployment of our method, we design our recommendation system not to use the context information from the other users with centralized system. We note that the number of available trainable personal data would be minimal as we limit the volume of training data, thus it can induce lack of generalization and local recommendation problem. However, we suggest that this can be disregarded in our environment as we aim the recommendation of repeated app use as mentioned earlier. We show that our proposed method can more accurately predict the repeatedly used apps than compared methods in Section 5.

2.2 Process Overview

Data cleaning. Prior to any recommendation process, the app recommendation system removes unnecessary app usages, as explained previous section. Examples of unnecessary apps include (i) *System Services*; The system service manages the core functions of devices (e.g. cellular, Wi-Fi, hardware sensors). For example, the package 'com.android.nfc' provides access to near field communication (NFC) functionality of the device. (ii) *Launcher*; The launcher app manages the default home screen in Android smartphone. Many of real Android users have installed customized launcher apps, so the app recommendation system should pay attention to such launcher apps. (iii) *System Applets*; System applets such as the settings menu and Wi-Fi connection manager are executed in the same way as the user apps.

Feature extraction. The app recommendation system uses context information to characterize app usage, either explicit or implicit one. Each piece of context information is extracted at every instance of app usage and is considered as a feature. To comprise multiple different context information about app usage, all context information is concatenated to construct a feature vector that lies in multi-dimensional space with a label.

In addition to the explicit information like sensor readings, we focus on the historical sequence of app usage as implicit information. On the mobile device, the user activity is recorded as a series of context data in order of time. Thus, these contexts are found to be sequential and dependent to the adjacent one in many

cases [9]. Similarly, the sequential behavior of app usage can be easily observed since users often use multiple apps to finish a specific task [8]. For example, suppose that there exists an Instagram user who loves taking and sharing photos. Then, we can easily infer that they often take several pictures with the camera app, then they open the Instagram app to share those pictures. Moreover, since the user would take and share pictures on a regular basis, the pattern of using sets of mobile apps such as the camera and Instagram would appear multiple times a day [16]. We introduce an App-Usage Tracking Feature (ATF) in the feature construction to represent such sequential trace of app usage. The formal definition and detail of each feature including ATF will be given in Section 3.

Learning. The main idea of our app recommendation system is that if the constructed feature vectors are similar, then their labels are probably the same. For example, suppose the existence of a person who checks his e-mail at his office every day at 10 o'clock. Then, it is natural to recommend his e-mail app around that time and location, so it should be reasonable to assume that the same app is executed in similar circumstances.

We assess the similarity between two instances of app usage by the 1^2 -norm distance. Therefore, distance becomes the most critical measure in our app recommendation scheme. However, since each feature may have a different influence in app execution, the distance measure can be ineffective in the raw feature space. Thus, metric learning should be performed before evaluating distances, in order that the distance measure should reflect the importance of each feature in such a way to scale each dimension properly.

To that end, we apply a Large Margin Nearest Neighbor (LMNN) [25] algorithm to learn a global linear transformation of multi-dimensional input space in a supervised way. It can be viewed as learning a Mahalanobis distance metric from the labeled examples. LMNN algorithm places feature vectors with the same label as close together as possible, and with different labels as far apart as possible to create a large margin between disparate labels.

Recommendation. We predict R apps, which are the most likely to be used next and recommend them to the user. As explained before, our main idea is that if the constructed feature vectors are similar, then their labels are probably the same. In other words, the current context information often influences the execution of apps.

Considering this idea, we use a k-Nearest Neighbor (k-NN) classification on app prediction, which is generally used for various classification and recommendation tasks and also successfully used in a number of previous works [16-17, 20]. We choose the k-NN as our app prediction method with following considerations: (i) k-NN does not requires assumptions about the user's app preference model and (ii) k-NN requires minimal number of parameters such as k and the number of features. Modification of them and interpretation of their effects are also trivial. These enables the personalization of our recommendation system with ease. (iii) The computational cost of learning process on k-NN is also minimal. Therefore, the resulting recommendation system can be trained on the mobile devices without central systems. We also compared some other classifiers such as Naïve Bayes and Decision Tree, but the k-NN showed the best prediction accuracies on our pre-experiments. Thus, we use the k-NN as a classification algorithm.

The recommendation is made by extracting features from the current context information first and linear-transforming the feature vector into the learned metric to apply the k-NN algorithm. Since we make weighted majority voting on classification, similarity as a weight is also measured by the Mahalanobis distance between feature vectors with the distance metric obtained from the learning phase. We select R candidates with the highest score on the recommendation.

We will explicate a detailed description of the learning and recommendation algorithm on Section 4.

3 Feature Construction

Now we describe the types of context information that are used in the feature construction process. Any explicit information like sensor readings or environmental variables can be the context information with an influence on app execution. Additionally, we introduce ATF to represent the implicit information of app usage history in multi-dimensional data.

3.1 Explicit Information

We exploit explicit information in smartphones that can be digitalized by a logger program. This can be extracted either from the hardware sensors (e.g. GPS, temperature) or the internal status of the mobile phone (e.g. current time, Wi-Fi connection). Some examples of explicit information used in our work are described in Table 1.

Table 1. Descriptions on explicit user context information selected as the main features

Feature	Description	e.g.
Time	The hour of time on app launch	18
Location	Latitude and longitude from GPS	(50.6, 3.4)
Weekend	The app is launched on weekend	1
AM	The app is launched in the morning	0
BTHset	A Bluetooth headset is connected	1
Headset	A non-BT headset is connected	1
PlugOn	The device is charging the battery	0
WIFI	Wi-Fi connection is established	0
Battery	The current battery level in decimal	57

In our system, each feature is normalized to a value in [0, 1] and vectorized appropriately if necessary. For example, the time feature has a scalar value in [0, 24).

Considering the periodicity of time, we embed it into a circle in a 2-dimensional space to connect the time before and after midnight as follows

$$\mathbf{x}_{\text{time}} = [0.5 + 0.5\sin(2\pi(hour/24)), \\ 0.5 + 0.5\cos(2\pi(hour/24))]. \quad (1)$$

For every app usage, we concatenate these features into a feature vector.

3.2 App-Usage Tracking Feature (ATF)

In this section, we introduce the App-Usage Tracking Feature (ATF), which captures implicit sequence patterns about previous app launches. As described in Section 2.2, list of recently used apps are useful for predicting the next app, since recently used apps would be highly related to the current app. We exploit such patterns in our prediction by capturing the sequence information as a feature.

We construct ATF, $\mathbf{x}_{\text{ATF}} \in \mathbb{R}^h$ as an h-dimensional feature vector, where h is the number of installed apps on a user's smartphone. We then calculate the feature from the history of previous $w \in \mathbb{N}$ app launches, where w is the size of the *window* which represents how many previous apps are considered when updating ATF. The i-th element of ATF $\mathbf{x}_{\text{ATF},i}$ represents a value assigned to App_i, which is calculated as

$$\mathbf{x}_{\text{ATF},i} = \sum_{j-1}^{w} r^{j-1}\mathbf{I}_{ij}, \quad (2)$$

where $\mathbf{I}_{ij}$ is defined by

$$\mathbf{I}_{ij} = \begin{cases} 1 & \text{if App}_i \text{ is } j-\text{th recetly used app} \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

Here, $r(0 < r \le 1)$ is the decay rate over time which is assigned to each previous app launch. We consider the decay rate since the older app launches may be less significant than the newer ones.

The example of calculating ATF is illustrated in Figure 1. Suppose that a user is now using Instagram app and recently used Camera and Maps app. When $w = 2$, the ATF calculation algorithm checks up to the 2nd recently used apps, so the 3rd or older app launches are not used to calculate the ATF. The $x_{ATF,\,Camera}$ for the latest launched app, Camera, is calculated as $r^{1-1} \cdot 1 = 1$ and the $x_{ATF,\,Maps}$ for Maps is $r^{2-1} \cdot 1 = 0.5$ when $r = 0.5$. The other components of $\mathbf{x}_{\text{ATF}}$ including $x_{ATF,\,Instagram}$ are 0.

By the definition of ATF, it stores what apps are used within w previous app usages from the current time. For an extreme example, if the two instances of app usage sequences are the same, then both ATFs capturing that sequence must be same by the equation. Similarly, two ATF vectors are to be computed and

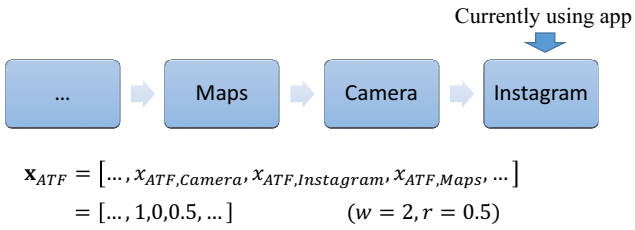


Figure 1. ATF example

embedded closely in the data space if both sequential app usages are similar in general cases. Such computation is in accord with our intention since the l^2-norm distance measure should capture how similar the app usage sequences are. Another important characteristic of ATF is that ATF is efficient in term of space complexity. Suppose that there are total h apps on the user's mobile phone. Then the total number of possible app sequence patterns within w time window is h^w, which is very large and practically intractable for computation when $w \ge 2$. The essential idea of our ATF is that we project this high dimensional h^w patterns into a low dimensional space (h dimension) by computing the occurrence count of each app with the decay rate. In addition, the calculation time of ATF is also computationally cheap. The time complexity of calculating the ATF is $O(w)$. In other words, it only depends on size of the window and the calculation of ATF can be easily done even on the mobile devices.

Note that ATF is treated as a feature like each context information, so ATF is also concatenated into the feature vector. Therefore, the feature vector is constructed as $\mathbf{x} = [\mathbf{x}_{\text{Time}}, \mathbf{x}_{\text{Location}}, \dots, \mathbf{x}_{\text{ATF}}]$.

4 Recommendation Algorithm

In this section, we present descriptions on implementing the learning and recommendation phase in our recommendation algorithm.

4.1 Learning

In the learning phase, we obtain and aggregate the training data point $(\mathbf{x}; y)$ which represents a single instance of app usage. The data point is calculated in every execution of app. Then we perform the distance metric learning to improve the accuracy of recommendation.

Metric learning. Our raw features have various ranges of values. For example, the value of Battery feature varies from 1 to 100 whereas IsAM is a binary feature. In addition, the significance of each feature differs for each user; if a user rarely uses Wi-Fi connectivity, the WIFI feature would have little meaning for prediction. In this case, using the raw distance measure obtained from the original feature data may not be effective [26]. Hence, we apply a distance metric learning which can

improve the prediction. In our system, we used well-known Large Margin Near Neighbor (LMNN) [25] method.

The LMNN maps training data onto a new metric space where each training data point $\mathbf{x}_i$ are placed close to the target neighbors $\mathbf{x}_j$, which have the same label as $\mathbf{x}_i$, whereas it is separated from the impostors $\mathbf{x}_l$ that are differently labeled to $\mathbf{x}_i$. This intuition is formulated as a loss function as follows:

$$\in(\mathbf{L}_{\mathrm{N}}) = (1-\mu) \in_{\text{pull}} (\mathbf{L}_{\mathrm{N}}) + \mu \in_{\text{push}} (\mathbf{L}_{\mathrm{N}}), \quad \textbf{(4)}$$

$$\in_{\text{pull}} (\mathbf{L}_{\mathrm{N}}) = \sum_{i,j} || \mathbf{L}_{\mathrm{N}} (\mathbf{x}_i - \mathbf{x}_j) ||^2, \quad \textbf{(5)}$$

$$\in_{\text{push}} (\mathbf{L}_N) = \sum_{i,j,l} (1-y_{il})[1+ || \mathbf{L}_N (\mathbf{x}_i - \mathbf{x}_j) ||^2 - || \mathbf{L}_N (\mathbf{x}_i - \mathbf{x}_j) ||^2]_+, \quad \textbf{(6)}$$

where μ is a weighting parameter and y_{ii} is 1 if and only if $y_i = y_l$, or 0. The $z_+ = \max(z, 0)$ denotes the standard hinge loss. In $\in_{pull} (\mathbf{L}_{\mathrm{N}}), \Sigma_{i,j}$ is a summation over the label pairs (i, j) having the same label. In $\in_{pull} (\mathbf{L}_{\mathrm{N}})$, the summation is over the label triples (i, j, l) so that differently labeled l data points could be placed away.

To minimize the loss function, the squared distance in the above term is substituted by a Mahalanobis distance metric $(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{M} (\mathbf{x}_i - \mathbf{x}_j)$ where $\mathbf{M} = \mathbf{L}_{\mathrm{N}}^{\mathrm{T}} \mathbf{L}_{\mathrm{N}}$, then the loss function is minimized by a semidefinite programming (SDP), thus we can compute the $\mathbf{L}_{\mathrm{N}}$ in a polynomial time.

4.1 Recommendation

As stated earlier, our main idea on app recommendation is that close data points would probably have the same app label. Thus, we make a recommendation by searching for the closest data points from the training data by *k*-NN classification [27].

We first compute the feature $\bar{\mathbf{x}}$ based on the current state of mobile device via the same process as learning. Then, we find the k neighbors of $\bar{\mathbf{x}}$, $\mathbf{x}_i$ from the training data, which are the closest data points to current feature $\bar{x}$. The distance between data points is measured as the 1^2-norm with the distance metric from LMNN as:

$$\text{dist}(\bar{\mathbf{x}}, \mathbf{x}_i) = || \mathbf{L}_{\mathrm{N}} (\bar{\mathbf{x}}, \mathbf{x}_i) ||^2 . \quad \textbf{(7)}$$

If the metric learning is omitted (naive learning), we set $\mathbf{L}_{\mathrm{N}} = \mathbf{I}$, then it becomes the Euclidean distance between the two raw feature vectors. After we find the k nearest $\mathbf{x}_i$ s, we assign the similarity between the $\bar{x}$ and each $\mathbf{x}_i$ as follows:

$$\text{sim}(\bar{\mathbf{x}}, \mathbf{x}_i) = \frac{1}{\text{dist}(\bar{\mathbf{x}}, \mathbf{x}_i) + \varepsilon} \quad \textbf{(8)}$$

Note that ε $(0 < \varepsilon < 1)$ is a small constant which is added to the denominator to prevent division by zero. We set ε as 0.1 in our implementation.

To predict the next app $\bar{y}$, our algorithm first computes the scores of each candidate app y as:

$$\text{score}(\bar{\mathbf{x}}, y_i) = \sum_{i \in k-NN} \text{sim}(\bar{\mathbf{x}}, \mathbf{x}_i) \cdot \delta(y, y_i), \quad \textbf{(9)}$$

where $\delta(y, y_i)$ is a Kronecker delta function, which is 1 if y_i is equal to y, and 0 otherwise. Then we predict $\bar{y}$ by selecting R apps that have the highest $\text{sorce}(\bar{\mathbf{x}}, y)$. If the number of apps having nonzero $\text{sorce}(\bar{\mathbf{x}}, y)$ is less than R, we draw a prediction only with apps having nonzero $\text{sorce}(\bar{\mathbf{x}}, y)$ and we do not make the further prediction.

5 Experiments

We evaluated the proposed method on a real world dataset collected from mobile phone users. We analyzed the various characteristics of our method, including the effectiveness of each feature, and compared the performance of our algorithm to other state of the art methods.

5.1 Setup

Dataset. The dataset was obtained from the Android mobile phones of 121 volunteers. To record each user's activities, we developed a special logger program and we asked all volunteers to install the program and use their mobile devices normally. The logger program records each user's app launches, sensor readings and other activities. The datasets were recorded for 3 months and we asked the volunteers to submit their recordings. After we collected all the data, we converted it as MATLAB format and performed the data cleaning process as described in Section 2. On the collected dataset, there were 430,988 activities in total. After the data cleaning process, we only used 77,698 records, 642 per user on average. The number of unique apps on the dataset was 795 after we cleaned the data and each user installed 40 apps on their mobile device on average.

Method. For each user, we separated the dataset into two parts by the time when each activity had been recorded. First 80% of the dataset (older) for each user was used as the training data, and the remaining 20% of the dataset (newer) was used as the test data.

As we designed a personalized system, we computed

the number of neighbors, k for each user separately, which is learned from the training data of that user only. We performed the distance learning via LMNN for each user separately as well. We conducted experiments on a Windows PC and we implemented all algorithms on this evaluation in MATLAB R2015b.

Metrics We measured two performance metrics to evaluate the recommendation accuracy; The weighted average of the recall and the DCG. Detailed descriptions are given as follows:

- **Recall.** The recall value is the direct performance metric of the recommendation accuracy in our test, since users only execute one app on each test. We followed the definition of recall stated in [28]. In this case, recall for a single test can assume either the value 0 (in the case of a miss) or 1 (in the case of a hit). When we calculate the recall value for each user u, we use the average value of all test cases on user u : $\mathrm{Re}call_U = \#hits_u / \#tests_u$
- **Discounted Cumulative Gain (DCG).** DCG is commonly used in information retrieval and also in recommendation systems to evaluate the quality of ranking generated by a prediction algorithm [17, 29-30]. In DCG, the relevance of the recommended item is either 0 or 1 in our case, since a user only picks one app or none of the candidates. It is discounted by the location at which the hit occurs, thus the DCG of each test case is calculated as $DCG = \log_2(j+1)$ if the recommendation is hit at j-th app among the R recommendations, or 0 otherwise. Similar to the recall, we can calculate the DCG value of each user u as the average of all test cases: $DCG_u = \Sigma\, DCG / \#tests_u$. If $R = 1$, the DCG is the same as the recall value.

In each evaluation, the accuracy score is calculated as the weighted average of the recall or the DCG score, weighted by the number of each user's test cases:

$$Acc = \frac{\Sigma_u \#tests_u \cdot Acc_u}{\Sigma_u \#tests_u}, \qquad \textbf{(10)}$$

where the Acc is either the $Recall$ or the DCG.

- **Compared methods.** We compared our proposed method using ATF and other features listed in Table 1 with a number of previous methods for an app prediction:
- **CLL.** Conditional Log Linear (CLL) is a discriminative model that represents a conditional probability distribution of app candidates given context information [16]:

$$P(y \mid \theta, \mathbf{z}) = \frac{\exp\{\theta^T \mathbf{f}(\mathbf{z}, y)\}}{\Sigma_{y'} \exp\{\theta^T \mathbf{f}(\mathbf{z}, y')\}}, \qquad \textbf{(11)}$$

where θ is a weight vector and $f(z, y)$ is the binary feature function which has its own target value and its output represents whether the given context information z meets the target.

- **KAP.** Liao et al. [17] introduced Implicit Feature (IF) on their k-NN based App Prediction (KAP) as a representation of app transitions. IF is constructed from app usage graph (AUG), which models the probability distribution over app transitions given previous app usage and transition interval among apps. IF is combined with features computed from other context information and used to k-NN classification to predict the next app.
- **Most Frequently Used (MFU).** MFU method counts every execution of each application and suggests applications in decreasing order of usage count.
- **Most Recently Used (MRU).** MRU method suggests recently executed applications from the most recent to the least recent ones. Both MFU and MRU methods have been used as baselines for app recommendation systems in previous studies.

5.2 Results

ATF parameters setting. We analyzed the impact of varying ATF parameters on prediction accuracy. The ATF has two parameters, the size of the window w and decay rate r. We tested with $r \in \{0.5, 0.6, ..., 1\}$ and $w \in \{1, 2, ..., 10\}$ and $R = 1$. The results are shown in Figure 2. The ATF with $w = 2$ showed the best recall for all r values. This suggests that considering 2 recently used apps on calculating the ATF is the most effective. It is interesting that treating previous apps with the same significance ($r = 1$) was the most effective when $w = 2$ and considering more previously used apps with higher decay rate r did not exhibit the improvement of accuracy. Since $(w, r) = (2, 1)$ showed the best results, we used these parameters for further experiments.

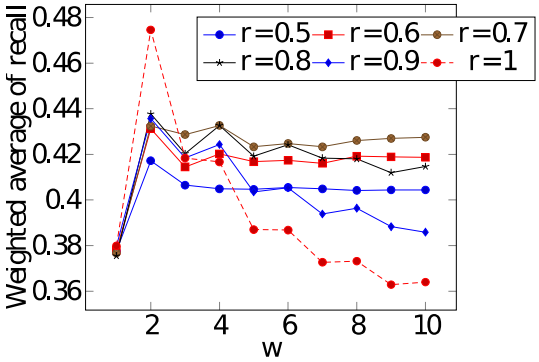


Figure 2. Impact of w and r on ATF on $R = 1$

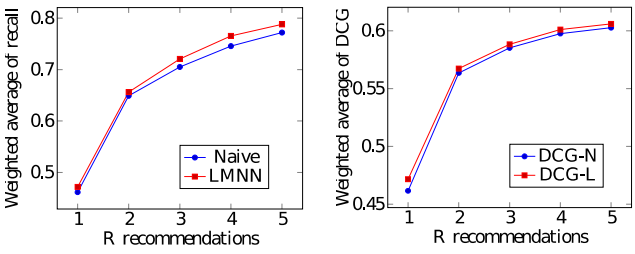
Impact of each feature. To analyze the effectiveness of each feature, we tested our algorithm using all possible combinations of explicit features listed in Table 1, and ATF (about 1000 cases). Then we picked the best performing feature combinations in recall. The left four results in Table 2 are the highest 4 recall values from all tested feature combinations.

Table 2. The highest 4 accuracy scores and their feature selections (left 4 columns) and the highest 2 results without ATF (right 2 columns)

Features $R=1$	ATF	Battery ATF	BTHset ATF	Location Battery ATF	Time WIFI Weekend	Location AM PlugOn
Recall	0.4726	0.4693	0.4663	0.4658	0.3147	0.3137
Features $R=5$	Location ATF	ATF	BTHset ATF	Weekend ATF	Location Headset WIFI	Location AM BTHset
Recall	0.7884	0.7880	0.7874	0.7870	0.6875	0.6848
DCG	0.6020	0.5996	0.5986	0.5985	0.4662	0.4702

Results in Table 2 shows that ATF is the most useful feature to next app prediction. The ATF showed the most and second highest recall accuracy for $R=1$ and $R=5$ without additional features. In this experiment, ATF is shown to be powerful enough for the recommendation on our dataset. Note that the importance of each features or each combination of features could be different on other mobile app usage data.

Impact of distance metric learning. In Figure 3, We present the impact of LMNN distance learning on prediction accuracy using ATF. As shown in Figure 3, the LMNN consistently improves the performance. The recall is improved from 0.472 to 0.462 for $R=1$ and 0.772 to 0.788 for $R=5$. The DCG accuracy is also improved from from 0.602 to 0.605 for $R=5$. Note that we limited the maximum number of iterations on optimization process of LMNN to 50, which is much smaller than the default setting of 1,000. Although the LMNN objective may remain suboptimal in this case, we found that the gain of accuracy is almost the same. We suggest that the practical implementation of distance learning on mobile device would be possible by limiting the maximum number of iterations.



(a) and DCG (b) accuracy using ATF

Figure 3. Impact of LMNN metric learning on recall

Comparison with other algorithms. In this section, we present the comparative results between the various recommendation algorithms stated earlier. We tested our algorithm using ATF only (A1) since the ATF, which is calculated from sequence of previously used apps, was the most effective single feature for recommendation on our dataset as shown as the results in Table II. In the same way, the features on most and second-most recently used apps (f_1, f_2, f_7, f_8) are used in CLL and IF is used in KAP. For comparison with A1, we also tested our method without ATF (using non-ATF context information only) noted as A2. The *k*-NN recommendation algorithm with LMNN distance learning described in Section IV is applied to A1, A2, and KAP methods.

The results of accuracy scores with respect to the values of R and boxplots of accuracy scores over all users at $R=3$ are listed in Figure 4. A1 showed the highest accuracy in every R recommendations, yielded recall accuracy of 0.472 at $R=1$, 0.788 in recall and 0.599 in DCG at $R=5$. On the other hand, CLL showed comparable accuracy with A1. Its performance was consistently inferior compared to our algorithm though. (recall=0.358 at $R=1$, recall=0.692, DCG=0.531 at $R=5$) For KAP with IF, it exhibited a lower accuracy than both A1 and CLL (recall=0.316 at $R=1$, recall=0.583, DCG=0.432 at $R=5$). The A2 showed a bit lower accuracy compared to CLL (recall=0.314 at $R=1$, recall=0.687, DCG=0.466 at $R=5$); however, it outperformed the baseline methods (MFU, MRU).

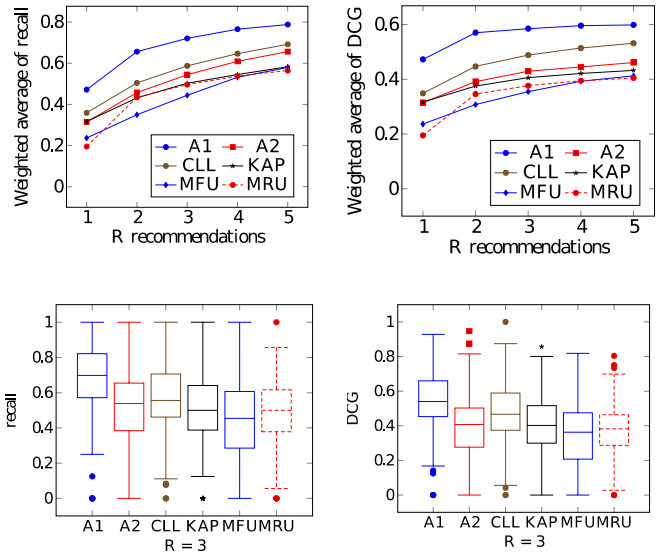


Figure 4. Accuracy scores of compared methods (top) and boxplots of user accuracy scores at $R=3$ (bottom)

The comparative experiment showed that our proposed algorithm with ATF outperforms both CLL and KAP which use information of previously used

apps. In addition, our method also presented highest median and first and third quartiles of accuracy scores over all tested users as shown as boxplots of accuracy scores. This suggests that the proposed system with ATF consistently provides superior recommendation for all users compared to other methods.

Comparing with KAP, it is also worth noting that our proposed method uses the same *k*-NN classification, however our method with ATF (A1) outperformed the KAP with IF. This shows that the ATF is more suitable feature for *k*-NN than IF to achieve higher performance. The ATF is also more efficient than IF on feature calculation time since the ATF calculation algorithm does not iteratively calculate feature vectors as IF algorithm.

Performance analysis by App categories. We divided the average recommendation accuracy results of 3 compared algorithms (A1, CLL and KAP) on $R=1$ by app categories. We aggregated the recommendation results of all user's test sessions first then we selected apps which have at least 50 launches per each app. Then we grouped each apps into 5 categories: (i) *Entertainment*; The entertainment apps are including game apps, music/video players, apps for watching web comics, camera, and photo-editing apps, etc. (ii) *Informative*; The informative apps are such as web browsers, weather and map apps and apps for online shopping, etc. (iii) *Messengers*; Text messengers including mobile messenger apps and short message service (SMS) apps. (iv) *Productivity*; The productivity apps are including e-mail clients, apps for mobile office, etc. (v) *Social Networks*; Apps for an access to social network services (SNS). The recommendation accuracy for each category is calculated as: $acc_c = \frac{hit_c}{test_c}$, where hit_c and $test_c$ are the number of hits and app launches corresponding to each app category. Table 3 summarizes the results. Note that Table 3 (d) is the distribution of app categories by total number of app launches.

The accuracy scores vary with app categories. For all algorithms, messenger and informative apps have 1st and 2nd highest recommendation accuracy, whereas accuracy of entertainment apps never exceeds 0.3. Increment of accuracy using A1 over other methods also varies with app categories. For example, the recall score of A1 on informative and messenger apps received the highest gain, which are the types of apps have the 2nd and 1st highest number of executions, respectively.

These results show that the ATF effectively predicted more frequently used type of apps such as messengers and informative apps. This behavior meets with our intention in overall system and ATF feature setup. Results from all 3 algorithms also suggest that it is effective to utilize the information about previously used app to predict frequently and repeatedly used apps since all of them showed better performance on more frequently used type of apps.

Table 3. Recommendation accuracies of compared algorithms by app categories ($R=1$)

(a) A1

Categories	acc_c
Entertainment	0.279
Informative	0.665
Messengers	0.814
Productivity	0.414
Social Networks	0.360

(b) CLL

Categories	%
Entertainment	0.240
Informative	0.465
Messengers	0.632
Productivity	0.314
Social Networks	0.315

(c) KAP

Categories	acc_c
Entertainment	0.166
Informative	0.451
Messengers	0.567
Productivity	0.316
Social Networks	0.289

(d) Distribution of app categories

Categories	%
Entertainment	9.2
Informative	24.7
Messengers	38.2
Productivity	16.6
Social Networks	11.3

6 Conclusion

In this paper, we proposed a mobile app recommendation system that predicts the next app to be used. We designed a novel feature, ATF, to incorporate the individual user's app usage behavior into the recommendation with a variety of context features, so that the user's behavior can be predicted more precisely. Based on these features, we adopt *k*-NN classification and LMNN metric learning to predict the next app accurately. We aimed to collect and learn only the target user's mobile data to ensure that the user privacy would not be compromised by our recommendation system. Our analysis on real world mobile data demonstrated that ATF has decent explanatory power on mobile app usage behavior and validated that our proposed method outperforms the other approaches, including the baseline methods such as MFU and MRU. We conducted further experiments to study the impact of various parameter settings and the behavior of recommendation methods among app categories.

The future work would be dealing with the situation when the training data has accumulated over a long time such as several months. As user behavior on app usage changes over time, the old learning data points would be less meaningful on the recommendations, thus we may need to consider the time when the training data is computed in the recommendation phase. The cold-start problem also need to be addressed in the future as our current method cannot provide recommendation when the training data has not been accumulated.

Acknowledgements

K. Jung is with Automation and System Research Institute (ASRI), Seoul National University. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (*MSIT) (No. 2016R1A2B2009759). This work was also supported by the Brain Korea 21 Plus Project from 2016 to 2018. We thank the anonymous reviewers for their thoughtful and constructive comments

References

[1] Supermonitoring, S*tate of Mobile 2013 (info-graphic)*, http://www.supermonitoring.com/blog/state-of-mobile-2013-infographic, 2013 [Online; accessed 16-Jul-2018].

[2] Nielsen, Smartphone: So Many Apps, *So Much Time*, http://www.nielsen.com/us/en/insights/news/2014/smartphones-so-many-apps--so-much-time.html, 2014 [Online; accessed 16-Jul-2018]

[3] B. Zhang, N. Wang, H. Jin, Privacy Concerns in Online Recommender Systems: Influences of Control and User Data Input, *Symposium on Usable Privacy and Security (SOUPS)*, Menlo Park, United States, 2014, pp. 159-173.

[4] H. Cao, M. Lin, Mining Smartphone Data for App Usage Prediction and Recommendations: A Survey, *Pervasive and Mobile Computing*, Vol. 37, pp. 1-22, January, 2017.

[5] C. Zhang, X. Ding, G. Chen, K. Huang, X. Ma, B. Yan, Nihao: A Predictive Smartphone Application Launcher, *International Conference on Mobile Computing, Applications and Services (MobiCASE)*, Seattle, WA, 2012, pp. 294-313.

[6] J. Keshet, A. Kariv, A. Dagan, D. Volk, J. Simhon, *Context-Based Prediction of App Usage*, arXiv:1512.07851, December, 2015.

[7] A. Parate, M. Böhmer, D. Chu, D. Ganesan, B. M. Marlin, Practical Prediction and Prefetch for Faster access to Applications on Mobile Phones, *ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)*, Zurich, Switzerland, 2013, pp. 275-284.

[8] T. X. Yan, D. Chu, D. Ganesan, A. Kansal, J. Liu, Fast App Launching for Mobile Devices Using Predictive User Context. *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, Low Wood Bay, UK, 2012, pp. 113-126.

[9] B.Huai, E. Chen, H. Zhu, H. Xiong, T. Bao, Q. Liu, J. Tian, Toward Personalized Context Recognition for Mobile Users: A Semisupervised Bayesian HMM Approach, *ACM Transactions on Knowledge Discovery from Data*, Vol. 9, No. 2, pp. 10:1--10:29, November, 2014.

[10] H. Zhu, E. Chen, H. Xiong, K. Yu, H. Cao, J. Tian, Mining Mobile User Preferences for Personalized Context-Aware Recommendation, *ACM Transactions on Intelligent Systems and Technology*, Vol. 5, No. 4, pp. 58: 1-58: 27, January, 2015.

[11] S. Liu, X. Meng, Context-Aware Mobile Proactive Recommendation, *Journal of Internet Technology*, Vol. 16, No. 4, pp. 685-693, July, 2015.

[12] H. Verkasalo, Contextual Patterns in Mobile Service Usage, *Personal and Ubiquitous Computing*, Vol. 13, No. 5, pp. 331-342, June, 2009.

[13] M. Bohmer, B. Hecht, J. Schoning, A. Kruger, G. Bauer, Falling Asleep with Angry Birds, Facebook and Kindle: A Large Scale Study on Mobile Application Usage, *International Conference on Human- Computer Interaction with Mobile Devices and Services (MobileHCI)*, Stockholm, Sweden, 2011, pp. 47-56.

[14] C. Shin, J. Hong, A. K. Dey, Understanding and Prediction of Mobile Application Usage for Smart Phones, *ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)*, Pittsburgh, PA, 2012, pp. 173-182.

[15] C. Xiang, D. Liu, S. Li, X. Zhu, Y. Li, J. Ren, L. Liang, HiNextApp: A Context-Aware and Adaptive Framework for App Prediction in Mobile Systems. *IEEE Trustcom/BigDataSE/ICESS*, Sydney, Australia, 2017, pp. 776-783.

[16] J. Kim, T. Mielikinen, Conditional Log-linear Models for Mobile Application Usage Prediction, *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery (ECML PKDD)*, Nancy, France, 2014, pp. 672-687.

[17] Z. X. Liao, S. C. Li, W. C. Peng, P. S. Yu, T. C. Liu, On the Feature Discovery for App Usage Prediction in Smartphones, *IEEE International Conference on Data Mining (ICDM)*, Dallas, United States, 2013, pp. 1127-1132, 2013.

[18] R. Baeza-Yates, D. Jiang, F. Silvestri, B. Harrison, Predicting The Next App that You Are Going to Use, *ACM International Conference on Web Search and Data Mining (WSDM)*, Shanghai, China, 2015, pp. 285-294.

[19] D. Kamisaka, S. Muramatsu, H. Yokoyama, T. Iwamoto, Operation Prediction for Context-aware User Interfaces of Mobile Phones, *The Annual International Symposium on Applications and the Internet (SAINT)*, Seattle, WA, 2009, pp. 16-22.

[20] Y. Xu, M. Lin, H. Lu, G. Cardone, N. Lane, Z. Chen, A. Campbell, T. Choudhury, Preference, Context and Communities: A Multi-faceted Approach to Predicting Smartphone App Usage Patterns, *International Semantic Web Conference (ISWC)*, Sydney, Australia, 2013, pp. 69-76.

[21] P. Yin, P. Luo, W. C. Lee, M. Wang, App Recommendation: A Contest between Satisfaction and Temptation, *ACM International Conference on Web Search and Data Mining (WSDM)*, New York, NY, 2013, pp. 395-404.

[22] H. Zhu, H. Xiong, Y. Ge, E. Chen, Mobile App Recommendations with Security and Privacy Awareness, *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, New York, NY, 2014, pp. 951-960.

[23] I. Shklovski, S. D. Mainwaring, H. H. Skúladóttir, H. Borgthorsson, Leakiness and Creepiness in App Space: Perceptions of Privacy and Mobile App Use, *ACM Conference on Human Factors in Computing Systems (CHI)*, Toronto, Canada, 2014, pp. 2347-2356.

[24] H. Almuhimedi, F. Schaub, N. Sadeh, I. Adjerid, A. Acquisti,

J. Gluck, L. Cranor, Yuvraj Agarwal, Your Location Has been Shared 5,398 Times! A Field Study on Mobile App Privacy Nudging, *ACM Conference on Human Factors in Computing Systems (CHI)*, Seoul, Korea, 2015, pp. 787-796.

[25] K. Q. Weinberger, L. K. Saul, Distance Metric Learning for Large Margin Nearest Neighbor Classification, *Journal of Machine Learning Research*, Vol. 10, pp. 207-244, February, 2009.

[26] N. Shental, T. Hertz, D. Weinshall, M. Pavel, Adjustment Learning and Relevant Component Analysis, *European Conference on Computer Vision (ECCV)*, Antibes, France, 2002, pp. 776-792.

[27] T. Cover, P. Hart, Nearest Neighbor Pattern Classification, *IEEE Transactions on Information Theory*, Vol. 13, No. 1, pp. 21-27, January, 1967.

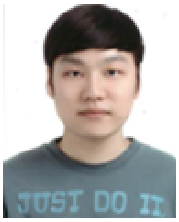
[28] P. Cremonesi, Y. Koren, R. Turrin, Performance of Recommender Algorithms on Top-n Recommendation Tasks, *ACM Conference on Recommender Systems (RecSys)*, Barcelona, Spain, 2010, pp. 39-46.

[29] K. Jarvelin, J. Kekalainen, Cumulated Gain-based Evaluation of IR Techniques, *ACM Transactions on Information Systems*, Vol. 20, No. 4, pp. 422-446, October, 2002.

[30] B. Liu, Y. Wu, N. Z. Gong, J. Wu, H. Xiong, M. Ester, *Structural Analysis of User Choices for Mobile App Recommendation*, arXiv:1605.07980, May, 2016.

Biographies

Yongkeun Hwang is a Ph.D. candidate in the ECE Dept. at Seoul National University. He received Bachelor of Science in Electronic Information Systems Engineering from Hanyang University in February 2014. His research interests are in the field of recommendation systems, and natural language processing.



Donghyeon Lee is a Ph.D. candidate in the ECE Dept. at Seoul National University. He received Bachelor of Science in Electrical and Electronic Engineering from Yonsei University in February 2012. His research interests are in the field of computer vision, and crowdsourcing.

Kyomin Jung is an associate professor in the ECE Dept. at Seoul National University. He received his PhD at MIT in 2009, and BSc at Seoul National Univ. in 2003 respectively. His main research areas include machine learning and natural language processing.