AppUsage2Vec: Modeling Smartphone App Usage for Prediction

Sha Zhao[1], Zhiling Luo[1], Ziwen Jiang[1], Haiyan Wang[1], Feng Xu[1], Shijian Li[1], Jianwei Yin[1], Gang Pan[2,1]

[1]*Department of Computer Science, Zhejiang University, Hangzhou, China*
[2]*State Key Lab of CAD&CG, Zhejiang University, Hangzhou, China*
{szhao, luozhiling, zjujzw, whyan, fxuzju, shijianli, zjuyjw, gpan}@zju.edu.cn

***Abstract*—App usage prediction, i.e. which apps will be used next, is very useful for smartphone system optimization, such as operating system resource management, battery energy consumption optimization, and user experience improvement as well. However, it is still challenging to achieve usage prediction of high accuracy. In this paper, we propose a novel framework for app usage prediction, called AppUsage2Vec, inspired by Doc2Vec. It models app usage records by considering the contribution of different apps, user personalized characteristics, and temporal context. We measure the contribution of each app to the target app by introducing an app-attention mechanism. The user personalized characteristics in app usage are learned by a module of dual-DNN. Furthermore, we encode the top-k supervised information in loss function for training the model to predict the app most likely to be used next. The AppUsage2Vec was evaluated on a dataset of 10,360 users and 46,434,380 records in three months. The results demonstrate the state-of-the-art performance.**

***Index Terms*—Smartphone applications, app usage prediction, user modeling**

I. Introduction

With the increasing prevalence of smartphones, the mobile application market has seen explosive growth in recent years, with Apple's app store having more than 2 million applications and Google's Android market also having above 3.8 million applications as of the first quarter of 2018 [1]. Applications (*Abbr.* apps) on smartphones can be considered as entries to access everyday life services such as communication, shopping, navigation, and entertainment. Users can easily download and install apps on their smartphones to facilitate their daily lives. As reported in [1], the average number of apps installed on a user's smartphone is around 56, and for some users, it is up to 150. Given the large number of installed apps and the limited screen size of smartphones, it is often tedious for users to search for the apps they want to use. In addition, for some apps such as the news apps it takes a while to download the latest content after they are turned on [2]. It is becoming quite an important issue that how to help users quickly find the apps they need and reduce the app load time. One effective way is to predict which apps will be used before one user actually needs them.

Predicting the next apps most likely to be used is very useful for smartphone system optimization, such as operating system resource management, battery energy consumption optimization, and user experience improvement. First, smartphones can pre-load apps into the memory for faster execution and remedy the launch delay by predicting the apps likely to be used in the near future. The apps that are running in the background but will not be used in the near future could be released, so that resources are allocated so as to optimize responsiveness subject to the finite resources available. Second, with the knowledge of the next app most likely to be used, the battery energy consumption can be planned in advance and optimized to improve energy efficiency. Generally, different apps drain battery energy at diverse speeds. For example, watching videos on smartphones drains the smartphone battery energy at a much faster speed. According to the predicted apps, the battery energy consumption can be planned to conserve the battery life of smartphones. Third, app usage prediction benefits the user experience improvement. Its immediate applications are in designing fast app launching UIs. In Example 1, we can use the recently used apps of 'eBay-WhatsApp-eBay' to predict PayPal, and then present the icon of PayPal on the main screen or highlight it, which can make it easier for the user to find the app he/she is likely to use next.

Example 1: *One app usage record of Tom is: 'eBay-WhatsApp-eBay-PayPal'. Tom is using 'eBay' for online shopping. Unexpectedly, he/she receives a message in WhatsApp. He/she checks the message and then returns to eBay and continues to shopping. After shopping, he/she pays with PayPal.*

There have been several studies on app usage prediction [3]–[8]. For example, Liao *et al.* [3] proposed a temporal-based model to predict the next app, achieving a recall of 80% at top 5 apps for 15 users. Huang *et al.* [6] predicted 38 users' app usage and achieved an accuracy of 69% at top 5 apps. Shin *et al.* [4] collected a wide range of contextual information and made personalized app prediction for 23 users, achieving an accuracy of 78% at top 5 apps. Natarajan *et al.* [5] modeled app usage sequences and achieved a recall of 67% at top 5 apps on a dataset of over 17,000 users.

Many approaches have been proposed to predict the next app. However, most existing approaches have some limitations. First, they treated the apps in a given historical sequence equally, and all the apps had the same contribution for the target app. Actually, app usage prediction can hardly be equally dependent on the used apps in a given sequence, especially

[1]https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

in the scenarios of drop-in apps. Drop-in apps refer to the apps are happening by chance when users are intentionally using other apps to serve a specific need, such as the drop-in WhatsApp in Example 1. Intuitively, the drop-in apps are of less significance to the target app. The existing approaches, like Markov chain and RNN (Recurrent Neural Network), can well model the app usage patterns in time sequence, such as eBay and PayPal being usually used together. However, it is difficult for them to capture users' intention in app usage especially when drop-in apps happen, and detect which app is more important for the target app.

Second, most existing approaches have not taken user personalized characteristics in app usage into account, such as one's preferences to each app and app usage patterns. An individual and apps are related to each other, since he/she uses apps depending on his/her preferences and needs [9]. Some approaches, such as Bayesian framework, assume users and apps to be independent from each other in terms of impacting a user's choice of next app [2]. Last but not least, a model for app usage prediction is required to be with a light-weight computational cost, since the goal of app usage prediction is to improve the user experience. In other words, the model can not only perform well for app usage prediction but also be a light-weight implementation.

In this paper, to address the problems mentioned above, we propose a novel framework for app usage prediction, AppUsage2Vec, inspired by the idea of Doc2Vec [10]. Our model is evaluated on a large scale dataset of 10,360 users in a time period of three months, and the results show its effectiveness and stability. The execution time of AppUsage2Vec in both prediction and training phase is investigated and compared with other approaches, showing that it is light-weight for app usage prediction and can be allocated to run on a client device. We train a generic (*i.e.* user-independent) model and individual models, respectively. The generic model performs much better than individual ones, so that it can provide potentially better prediction performance to address the cold-start problem, especially when each user's training data size is very small. The contributions of our paper are two-fold.

(1) We propose a novel framework for app usage prediction, called AppUsage2Vec, and propose a top k based loss function for optimization of training. It is validated by a large-scale real-world dataset and achieves the state-of-the-art performance.
(2) AppUsage2Vec embeds three types of features and combines them together for the prediction task: 1) app attention; 2) user personalized characteristics in app usage; and 3) temporal context of app usage.

II. Related work

In recent years, a number of studies have attempted to predict the next app that will be used on smartphones by applying machine learning algorithms. [1], [3]–[8], [11]–[19]. The studies predicted the next app by modeling app usage sequences and exploiting contextual information. This section will review the related work in app usage prediction.

Considering that the consecutive apps are dependent upon their immediate previous apps in the app usage sequence, it is often assumed that Markov property stands that the next app is only related to the latest app. The studies in [1], [5], [6], [11] therefore proposed to model the app usage sequences by using Markov models, both by a single hidden Markov model [1], [5], [6] and a mixture of Markov models [11]. As an example, Huang *et al.* [6] predicted 38 users' app usage by building up a first-order Markov model to learn the probabilities of switching any two apps in an app transition matrix, and achieved the prediction accuracy of 69% at top 5 apps. It was found, the correlation between any two apps in app usage sequence has a strong contribution to the prediction accuracy. Natarajan *et al.* [5] modeled the app usage sequences using a cluster-level Markov model, which segments app usage behaviors across multiple users into a number of clusters. The Markov model was trained on the collective data for each group of user to model the group-level app usage characteristics, and achieved a recall of 67% at top 5 apps on a dataset of over 17,000 users.

Liao *et al.* [1] learned the app usage features covering both app transitions and duration with the Markov properties. They built a temporal app usage graph for each user, in which the node and directed edge represented an app and a transition. The app usage features of app transition matrix were learned in the training phase of the Markov model. Then, the optimal combination of the next app and other features, such as duration, was jointly estimated in the prediction phase. Finally, kNN (k Nearest Neighbors) classification model was adopted to predict app usage, and achieved a recall of 80% at top 5 apps for 50 users. Parate *et al.* [11] proposed a variant of the Predicting Matching algorithm by making a combination of Markov models of different orders for app usage prediction, and achieved an accuracy of 81% at top 5 apps for 34 users. Although Markov models learn the correlation among sequentially used apps, they assumed the next app is only related to the latest app and ignored the multiple apps in the previous time steps.

In addition to Markov models, a Bayesian framework has been popularly applied to predict the app usage in many studies. It was mentioned that merely considering the dependencies between the app usage behaviors was not enough to make the accurate prediction of the next app because of the high dependency of app usage on contexts, such as time, location, and phone status (e.g., battery level, screen brightness) [20], [21]. Many studies combined various features for app usage prediction by using Bayesian methods. Among [4], [6], [15], [22], the Bayesian framework was used for app usage prediction by combining app usage history and contexts in a unified manner. For example, Huang *et al.* [6] predicted the user's app usage by using a Bayesian framework in combining various features, such as location, phone status (e.g., silent mode), and app usage features. The study suggested that three types of features can be combined using a Bayesian model, which estimated the probability of the next-app candidates by multiplying the posterior probabilities computed using each

feature. The Bayesian model achieved an accuracy of 69% for top 5 candidate apps.

Shin *et al.* [4] collected a wide range of contextual information in a smartphone, and made personalized app prediction based on a naive Bayes model for 23 users, achieving an accuracy of 78% at top 5 apps. Zou *et al.* [14] proposed Bayesian methods to predict the next app based on the app usage history, and achieved an accuracy of 86% at top 5 apps. Baeza *et al.* [7] combined app session features with contextual features in a parallelized Tree Augmented Naive Bayes to predict the next app for 38 users. The app session features were extracted from app usage records by using Word2Vec [23]–[25], which learned distributed representations of app sessions by considering the sequential relationships existing among app actions. Bayesian models combine app usage features and contextual features for app usage prediction, but, the features used are assumed to be independent from each other for the prediction task. Actually, the app usage is closely related to contextual information, such as time.

Apart from Markov and Bayesian models, Do et al. [13] also compared a few other models to predict the next app, such as linear regression, logistic regression, and random forest. Xu *et al.* [15] proposed a two-phase classification model to predict the app usage patterns. The two-phase model included a personalized classification phase, which aimed to group users with similar app usage patterns, and a community-aware prediction phase, which aimed to predict the next app. In the classification phase, the apps were classified in terms of the app bag, which was a set of the contextual features of using each app. The model was evaluated on a dataset of 4600 users, achieving an accuracy of 67% for top 5 candidate apps.

To summarize, there have been many studies to predict the next app most likely to be used. However, most of the above studies are limited by evaluating prediction models for relatively small populations, such as 23 users [4], 38 users [6] and 52 users [12]. A few studies were evaluated for relatively large populations [5], [15], however, the prediction performance was lower than 70%. Xu *et al.* [15] predicted 4606 users' next app and achieved an accuracy of 67% at top 5 apps. Natarajan *et al.* [5] achieved a recall of 67% at top 5 apps on a dataset of over 17,000 users. Furthermore, they did not consider the contribution of different apps to the target app, and neglected users' personalized characteristics in app usage, degrading the prediction performance.

III. Problem description

Apps on smartphones are used in time order. We treat a series of apps used in a certain period as a sequence. Apps are often used in conjunction with other relevant apps to serve one need. For example, if a user launches the 'contacts' app, the next app is likely to be the 'mail' or the 'message' app. The app that a smartphone user will use next intuitively depends on the sequence of recently used apps [5]. As also reported by Huang [6], the correlation between sequentially used apps has a strong contribution to the accuracy of app prediction. In this paper, we try to predict the app one user most likely to use next based on his/her n most recently used apps. In Example 1, one app sequence of Tom is - 'eBay-WhatsApp-eBay-PayPal'. Taking this sequence as an example, we use the first three apps of 'eBay-WhatsApp-eBay' to predict the last app, PayPal, from all his/her installed apps.

Definition 1 (App Usage Prediction): Given a set of apps $\mathcal{A}$ and an observed sequence consisting of n most recently used apps $(a^u_{r-n}, a^u_{r-n+1}, \ldots, a^u_{r-1})$, where each app $a \in \mathcal{A}$, user $u \in \mathcal{U}$, predict the next app a^u_r from $\mathcal{A}$ that the user u most likely to use.

Formally, we need to estimate the conditional probability:

$$Pr(\cdot|a^u_{r-n}, a^u_{r-n+1}, ..., a^u_{r-1}, u) \tag{1}$$

where a^u_{r-i} means the $(r-i)$-th app of the user u, ordered from the latest to the oldest.

Each app will be with a probability that reflects how likely it will be used next. We consider the app with the highest probability as the one most likely to be used next. It is straightforward to transform into a *top* k prediction problem by sorting apps according to the estimated probability scores.

There are some important factors for app usage prediction. First, the app most likely to be used next intuitively depends on the sequence of apps used recently [5]. The recently used apps are treated as atomic units and independent with each other in some methods, such as naive Bayes. However, the correlation between sequentially used apps has a strong contribution to the prediction accuracy [6], since apps are usually used together for the same task. Thus, the apps in a sequence should be combined together for app prediction, rather than treated independently. Second, users play an important role for app usage prediction. Users use apps depending on their preferences and needs. Taking such personalized characteristics in app usage into consideration is helpful for prediction. But, none of the existing models has explored user personalized characteristics for app usage prediction.

In order to make both app sequences and users contribute to the prediction task, we reduce Doc2Vec [10] to the problem of app usage prediction. Doc2Vec predicts the next word by exploring a paragraph and a word sequence in a given context in the paragraph. Specifically, every word is mapped to a unique vector, as well as each paragraph. Word vectors are averaged, concatenated, or summed as a feature vector that is concatenated with the paragraph vector for predicting the next word.

Taking the analogy to word and document modeling, we can treat each user as a document and each app as a word, to predict the app most likely to be used next. However, the Doc2Vec model has some limitations in app usage prediction.

1) *Each app in a given sequence is treated equally to the target app.* In the original Doc2Vec model, each word vector in a given sequence is treated equally and has the same contribution to the target word. Actually, the apps in a sequence have the different contribution for the target app, especially when drop-in apps happen. Intuitively, the drop-in apps, like the WhatsApp in Example 1, contribute

less than other apps in the sequence. Thus, it is required to detect which app is more important for the target app.

2) *User personalized characteristics in app usage are described in a simple way.* In the original model, the user vector is simply mapped to a linear distributed vector, which cannot well encode users' app usage patterns.
3) *App vectors and the user vector are combined by simple concatenation.* In Doc2Vec model, word vectors and the paragraph vector is simply concatenated, making them independent from each other. One user and used apps are related to each other, since the user uses apps depending on his/her needs and preferences. The simple concatenation can not reflect users' preferences to each app.
4) *Temporal context is not considered.* App usage is highly dependent on the time periods in one day [3]. App usage behaviors on smartphones have been shown to exhibit specific temporal pattern [26]–[28]. For example, SMS and Phone are shown to have an evenly distributed pattern, whereas apps like news or weather apps are used more frequently in morning hours. However, the original Doc2Vec model does not take temporal context into account.

IV. AppUsage2Vec

We propose a novel framework named as AppUsage2Vec for app usage prediction, which is inspired by the approach of Doc2Vec [10]. We start by introducing an overview of the framework, and then discuss how to overcome the limitations mentioned above in detail.

A. Overview

In AppUsage2Vec, every user u is mapped into a unique distributed vector v_u, represented by a row in matrix U and every app a is also mapped into a unique distributed vector v_a, represented by a row in matrix A. One user vector and n app vectors that are most recently used by the user are taken as input for predicting the next app. The sequence is of fixed-length with n apps and sampled from a sliding window over the user's app usage records. The user vector is shared across all sequences generated from the same user but not across users. The app vector matrix A, however, is shared across users, i.e., the vector for the app of 'WeChat' is the same for all users.

In order to overcome the limitations mentioned above, we add three components, including an app-attention mechanism, dual-DNN (Deep Neural Network), and temporal context. More specifically, we introduce an app-attention mechanism to measure the contribution of different apps for the target app. It decides which app we should pay attention to. It is straightforward that the app paid to more attention contributes more to the target app. By adopting the app-attention mechanism, a *sequence vector* is built for prediction, which is the weighted sum of the app vectors with different attention weight. Sequentially, we design a dual-DNN model to learn user personalized characteristics in app usage behaviors, so

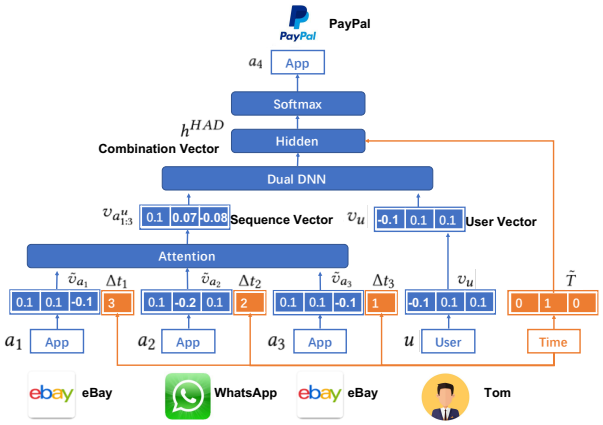


Fig. 1: Illustration of AppUsage2Vec Model.

that user vector and sequence vector are represented by automatically integrated deep features that can explicitly encode app usage patterns. In order to represent one's preferences to each app, we employ Hadamard product on the user and app sequence vectors. Finally, considering the high dependency of app usage on temporal context, time features of app usage is added to the model for the prediction task. Taking Example 1, we show the overview of AppUsage2Vec in Fig. 1.

More formally, given one user u and a sequence of n ordered apps $(a^u_{r-n}, a^u_{r-n+1}, \ldots, a^u_{r-1})$, the objective of the AppUsage2Vec model in the training procedure is to maximize the cross entropy

$$\frac{1}{|\mathcal{D}|}\sum_{r=n}^{|\mathcal{D}|} Pr(a^u_r|a^u_{r-n}, ..., a^u_{r-1}, u) \log Pr(a^u_r|a^u_{r-n}, ..., a^u_{r-1}, u) \tag{2}$$

where $\mathcal{D}$ is the set of all sequences. The prediction task is typically handled by a multiclass classifier. At prediction time, we employ softmax as activation, and we have

$$Pr(\hat{a}|a^u_{r-n}, a^u_{r-n+1}, ..., a^u_{r-1}, u) = \frac{e^{y_{\hat{a}}}}{\sum_i e^{y_i}} \tag{3}$$

Each of y_i is un-normalized probability for each predicted output app $\hat{a}$, computed as

$$y = W^T h(v_{a^u_{r-n:r-1}}, v_u) \tag{4}$$

where W is the parameter of softmax. $h(v_{a^u_{r-n:r-1}}, v_u)$ is combination vector, i.e., the concatenation of the sequence vector $v_{a^u_{r-n:r-1}}$ and the user vector v_u. The sequence vector $v_{a^u_{r-n:r-1}}$ is a weighted sum of all n app vectors $\{v^u_{a_{r-n}}, v^u_{a_{r-n+1}}, ..., v^u_{a_{r-1}}\}$.

The softmax function outputs the probability distribution over all candidate apps. The app that has the highest probability is considered as the one that most likely to be used next. The *top k* apps are selected by sorting apps according to the computed probability scores in a descending order.

In AppUsage2Vec, the app and user vectors are initialized randomly at the beginning and embedded with the same dimensionality. Both of them are trained using stochastic gradient descent and the gradient is obtained via backpropagation.

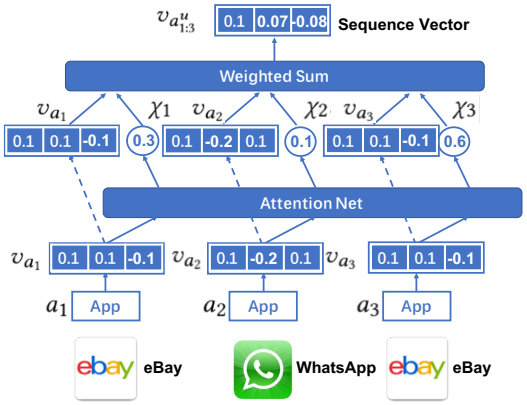


Fig. 2: Illustration of app-attention mechanism.

At every step of stochastic gradient descent, one can sample a fixed-length sequence from a random user, compute the error gradient from the network in Fig. 1, and use the gradient to update app vectors A, user vectors U, and softmax parameter W in the model.

B. App-Attention Mechanism: Measuring the Contribution of Each App

In the original Doc2Vec model, each app in a sequence is treated equally. It is a simple assumption that apps in the context are equally relevant to the target app. The app vectors are averaged to build the sequence vector $v_{a^u_{r-n:r-1}}$, shown in Eq. (5).

$$v_{a^u_{r-n:r-1}} = \frac{1}{n}\sum_{i=1}^{n} v_{a_{r-i}} \tag{5}$$

Actually, app usage prediction can hardly be equally dependent on all apps in the sequence. Apps have the different contribution to the target app. In the scenarios of drop-in apps, the drop-in apps are intuitively less of significance to the target app. In Example 1, Tom is interrupted by the message of WhatsApp when he is intentionally shopping with eBay. eBay is usually used together with PayPal for online shopping. Intuitively, eBay is more important than the drop-in WhatsApp for predicting PayPal. Thus, we should pay more attention to eBay rather than WhatsApp for the prediction of PayPal. It is worth noting that the oldest eBay is more important than WhatsApp, even it is earlier used than WhatsApp. It suggests that the contribution of each app is not completely dependent on its time order. It is necessary to capture users' intention in app usage and detect which app is more important for the prediction task.

In order to measure the contribution of each app to the target app, we introduce an app-attention mechanism. App-attention learns which app to attend to based on the sequence and what it has produced so far [29]. In particular, the order of each app in the sequence is also considered. The attention mechanism computes the weight of each app in the sequence for the target app, which makes each app have the different contribution. One app with a higher weight should be paid to more attention for the prediction task. The weights of all app vectors are summed to 1. With the app-attention mechanism, the sequence vector is built by a weighted summation of all the app vectors instead of an average operation used in the original model. By letting the model have an attention mechanism, we can infer users' intention from app usage sequences and further recognize which app is more important for the goal. With Example 2, the app-attention mechanism is illustrated in Fig. 2. It is worth noting that, even for the same app of 'eBay', the first and the third one are with different attention weights, due to their different time orders in the sequence. WhatsApp has the smallest weight for PayPal, although it is used later than the oldest eBay.

Example 2: *The distributed vectors of the apps {'eBay', 'WhatsApp', 'eBay'} in the sequence, are weighted as {0.3, 0.1, 0.6}, respectively by the app-attention mechanism, for predicting the target app of 'PayPal'. Three app vectors have different contribution to build the sequence vector, and the latest used eBay has the highest weight.*

Formally speaking, consider the app vectors $\{v_{a_{r-i}} | i \in 1, ..., n\} \subset \mathbb{R}^G$, where G, *embedding size*, denotes the dimensionality of v_a. The app vectors are integrated by a hidden layer. It involves an affine parameter $\boldsymbol{W}_v$ and a bias parameter $\boldsymbol{b}_v$, and gets the hidden vector $H_v \in \mathbb{R}^G$. Then, the attention probability $\boldsymbol{\chi}$ is the normalization on H_v. With the help of $\boldsymbol{\chi}$, the sequence vector is the weighted sum of all the app vectors. The process is illustrated by Eq. (6). Here, the superscript ATT is used to distinguish from the naive average app vector $v_{a^u_{r-n:r-1}}$ in the original model. The parameters are summarized as $\boldsymbol{\theta_{ATT}}$.

$$\begin{aligned} H_v &= tanh(\boldsymbol{W}_v[v_{a_{r-n}}, v_{a_{r-n+1}}, ..., v_{a_{r-1}}] + \boldsymbol{b}_v) \\ \boldsymbol{\chi} &= \frac{H_v}{\|H_v\|_1} \\ v^{ATT}_{a^u_{r-n:r-1}} &= \sum_{i=1}^{n} \chi_i v_{a_{r-i}} \end{aligned} \tag{6}$$

C. Dual-DNN: Learning User Personalized Characteristics

As mentioned above, users play an important role for app usage prediction. Users have different preferences to each app [30]. Taking such personalized characteristics in app usage into consideration is helpful for prediction. However, in the original Doc2Vec model, user vector is simply mapped to a linear distributed vector, which cannot well learn personalized characteristics in app usage. In order to overcome this limitation, in our AppUsage2Vec model, we design a dual-DNN module to learn user personalized characteristics in app usage sequence, shown in Fig. 3. More specifically, it consists of two parallel DNNs, one used on the user vector and another on the sequence vector. The user and app sequence will be represented by automatically integrated deep features that can explicitly encode app usage patterns.

Each DNN [31] is with fully connected layers with G neural cells and activated by $tanh$. The superscript a and u are used to mark the DNN for the app sequence and DNN for the user, respectively. A layer indicator i is marked as superscript, ordered from the shallow layer to the deep layer. For the layer

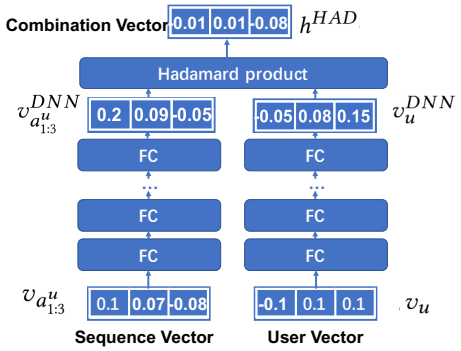


Fig. 3: Module of dual-DNN with Hadamard product.

$Z^{a,i}$, the i-th layer at DNN for the app sequence, an affine parameter $\boldsymbol{W}^{a,i}$ and a bias $\boldsymbol{b}^{a,i}$ are introduced. Note that all Z have the same dimensionality as v, i.e., $Z \in \mathbb{R}^G$. The parameters of dual-DNN for the apps and user are summarized as $\boldsymbol{\theta^a_{DNN}}$ and $\boldsymbol{\theta^u_{DNN}}$, respectively. Eq. (7) and Eq. (8) show how the user vector and sequence vector can be learned by dual-DNN.

$$\begin{aligned} Z^{a,1} &= tanh(\boldsymbol{W}_d^{a,1} v^{ATT}_{a^u_{r-n:r-1}} + \boldsymbol{b}^{a,1}), \\ Z^{a,2} &= tanh(\boldsymbol{W}_d^{a,2} Z^{a,1} + \boldsymbol{b}^{a,2}), \\ &\cdots \\ v^{DNN}_{a^u_{r-n:r-1}} &= tanh(\boldsymbol{W}_d^{a,L} Z^{a,L-1} + \boldsymbol{b}^{a,L}), \end{aligned} \tag{7}$$

$$\begin{aligned} Z^{u,1} &= tanh(\boldsymbol{W}_d^{u,1} v^{ATT}_u + \boldsymbol{b}^{u,1}), \\ Z^{u,2} &= tanh(\boldsymbol{W}_d^{u,2} Z^{u,1} + \boldsymbol{b}^{u,2}), \\ &\cdots \\ v^{DNN}_u &= tanh(\boldsymbol{W}_d^{u,L} Z^{u,L-1} + \boldsymbol{b}^{u,L}). \end{aligned} \tag{8}$$

A good combination of users and apps can be used as a predictive feature for the target app. However, in the original Doc2Vec model, the sequence and user vector are combined by simple concatenation. It is formally described as:

$$h(v_{a^u_{r-n:r-1}}, v_u) = [v_{a^u_{r-n:r-1}}, v_u] \tag{9}$$

where $[\,\cdot\,]$ represents the concatenation operation on vectors.

The simple concatenation operation makes the sequence vector and user vector independent with each other for predicting the next app. Actually, the user and sequence of apps are closely related to each other, since one user determines what apps to use usually depending on his/her needs. Moreover, users have different preferences to apps, and the apps in the given sequence have different contribution to the target app. Hadamard product is a point-wise multiplication. Employing Hadamard product on the user and app sequence vectors can not only build the close relationship between users and apps, but also represent users' preferences to each app for the prediction task. In AppUsage2Vec, we build the combination vector of the user and sequence vector by Hadamard product, as shown in Eq. (10).

$$h^{HAD}(v_{a^u_{r-n:r-1}}, v_u) = v^{DNN}_{a^u_{r-n:r-1}} \odot v^{DNN}_u \tag{10}$$

where $\odot$ is Hadamard product. h^{HAD} refers to the combination vector obtained by Hadamard product.

D. Temporal Context

A smartphone user uses apps depending on his/her context. The most common and widely used contextual information is time [2]. App usage behaviors on smartphones have been shown to exhibit specific temporal pattern [26], [27], [32]. It was also mentioned in [3] that merely considering the dependencies between apps was not enough to make accurate prediction of the next app simply because of the high dependency of app usage on the temporal context. However, the original Doc2Vec model does not take temporal context into account.

In AppUsage2Vec, we introduce time as a new feature. With time taken into consideration, our objective is to maximize the following conditional probability,

$$Pr(a^u_r | a^u_{r-n}, a^u_{r-n+1}, ..., a^u_{r-1}, u, \mathcal{T}_{a^u_{r-n}}, \mathcal{T}_{a^u_{r-n+1}}, ..., \mathcal{T}_{a^u_{r-1}}) \tag{11}$$

where $\mathcal{T}_a$ represents the start timestamp of using the app a.

We propose two policies to use time features. First, we take account of the time difference Δt of each app and the latest used app in the sequence, considering that the latest used app is an important predictor for the next app [4], [6]. Specifically, the time difference in minutes of the i-th app and the latest used app is calculated by $\Delta t_i = \mathcal{T}_{a^u_{r-1}} - \mathcal{T}_{a^u_{r-i}}$. The input app vector is updated by appending one dimension that describes Δt, as shown in Eq. (12). By doing this, the time difference can be considered in the app-attention mechanism to compute the sequence vector.

$$\tilde{v}_a = [v_a, \Delta_a] \tag{12}$$

Another policy to utilize time features is to take the timestamp of the latest used app as the current time for the prediction. More specifically, we encode the time into a one-hot *time vector* $\tilde{T}$ of 31 dimensions, in which one of 24 dimensions with value of 1 indicates the hour in one day, and one of 7 dimensions with value of 1 indicates the day in one week. The current time can be combined together with the user and sequence vector to the combination vector. In other words, the combination vector is updated by appending the time vector $\tilde{T}$, shown in Eq. (13).

$$\tilde{h} = [h^{HAD}, \tilde{T}] \tag{13}$$

E. Top k Based Training

In order to train the AppUsage2Vec model, we need to estimate the parameters, including W in softmax, $\boldsymbol{\theta}_{ATT}$ in the app-attention mechanism, $\boldsymbol{\theta}^a_{DNN}$ and $\boldsymbol{\theta}^u_{DNN}$ in dual-DNN. The parameters can be summarized as $\Theta = (W, \boldsymbol{\theta}_{ATT}, \boldsymbol{\theta}^a_{DNN}, \boldsymbol{\theta}^u_{DNN})$. The objective function of the model in the training phase is the negative cross entropy on

conditional probability of the target app with respect to an observed app sequence, as shown in Eq. (14).

$$\arg\min_{\Theta} -\frac{1}{|\mathcal{D}|}\sum_{r=n}^{|\mathcal{D}|} p_{a_r^u}\log p_{a_r^u} \tag{14}$$

where

$$\boldsymbol{p} = Pr(\cdot|a_{r-n}^u, a_{r-n+1}^u, ..., a_{r-1}^u, u, \mathcal{T}_{a_{r-n}^u}, \mathcal{T}_{a_{r-n+1}^u}, ..., \mathcal{T}_{a_{r-1}^u}) \tag{15}$$

With training the parameters, we can obtain top k apps by sorting the probability computed by softmax in Eq. (3). Obviously, it is already a good prediction if the target app is one of the top k apps when k is small. However, the optimization of the objective function still continues to improve the probability $p_{a_r^u}$, and even cause over-fitting. Motivated by this idea, we adjust the objective function by introducing hinge loss [33] on top k. Formally, it is described as

$$\arg\min_{\Theta} -\frac{1}{|\mathcal{D}|}\sum_{r=n}^{|\mathcal{D}|} \alpha^{\mathbf{1}(p_{a_{r-n}^u} > \boldsymbol{p}_{[k]})} p_{a_{r-n}^u}\log p_{a_{r-n}^u} \tag{16}$$

where $\mathbf{1}(\cdot)$ is the indicator function and $p_{[k]}$ denotes the k-th largest value of p, and α is discount coefficient. In this procedure, gradient descent is employed to optimize the objective function shown in Eq. (16).

V. Experiments

In this section, we conducted extensive experiments to show the effectiveness of AppUsage2Vec. We first described the dataset used in the experiments.

A. Experiment Setup

*1) **Dataset**:* We tested AppUsage2Vec with a large-scale real-world dataset of app usage log provided by a mobile Internet company, containing 10,360 smartphones from Zhejiang province, China, and spans three months from Aug. 23th, 2017 to Nov. 23th, 2017. There are 46,434,380 records in total with each one consisting of identification (ID) of each smartphone (anonymized), the start timestamp, and the user-agent field of the client. The privacy issues of the dataset are carefully considered. The dataset does not contain any personally identifiable information, where the "user ID" has been anonymized and does not contain any user metadata. All the researchers are regulated by the strict non-disclosure agreement and the dataset is located in a secure off-line server. To use the dataset for evaluation, we performed a *pre-processing* work:

- *Extracting app usage records*. From the dataset, we inferred app identity according to the name of user agent of the client by utilizing a systematic framework: SAMPLES [34]. We also performed some minor manual modifications. For example, for those user agents with different versions but for the same app, we merged their names. This method enabled us to accurately label most of the apps found in our dataset. We also manually verified its accuracy and checked the inferred apps. With app identity, we extracted app usage records, each of which consisting of smartphone ID, start timestamp, and app identity. 9,373 unique apps were inferred in total. Each user uses 27.65 unique apps in average in the three months.
- *Merging the consecutive usage records of the same app in one minute.* There are cases that an app is used multiple times in a short time period. In this case, the repeated app sessions of the same app are likely to be usage independent from other apps. We de-duplicated the repeated app records from the same app by merging its consecutive records in every one minute.
- *Segmenting the app usage records into a series of app usage sequences.* Two app usage records were grouped into the same sequence if the time gap between them is equal to or less than 7 minutes. According to our observation, for 74.79% of the records, the time interval between two consecutive records is less than 7 minutes. For the sequences with 3 or more apps, the average number of apps in each sequence is 8.35, the average duration is 10.17 minutes, and per user has 315.46 sequences in average during the three months.
- *Filtering.* We focused on users who used apps more frequently. 1,621 users were removed, whose number of sequences in the three months is smaller than 50. We also focused on apps that were used more frequently and we selected the top 2000 most frequently used apps.

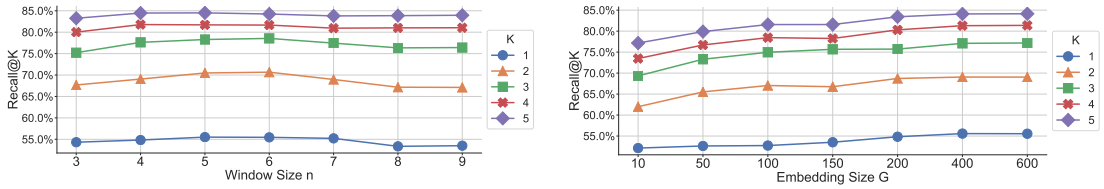
Overall, there were 8,739 users and 2000 frequently used apps used for experiments. The data of the first two months from Aug. 23 to Oct. 23, 2017 was used for training, and that of the last month was used for testing.

*2) **Performance measurement**:* We used the criterion of recall to measure the performance of AppUsage2Vec. The recall was computed when top k apps with the highest probability were selected, namely *Recall@*k. Recall@k was computed by Eq. (17) [1], [35]. Bigger value means better performance. We tested Recall@1, Recall@2, Recall@3, Recall@4, and Recall@5 in the following experiments. Our model was implemented in python with keras.

$$Recall@k = \frac{\sum_{i=1}^{|\mathcal{D}^{Test}|}\mathbf{1}(p_{a_i} \geq \boldsymbol{p}_{[k]})}{|\mathcal{D}^{Test}|} \tag{17}$$

*3) **Compared approaches**:* We selected the following algorithms to compare. We tested the traditional methods of MRU (Most Recently Used) and MFU (Most Frequently Used). We investigated the methods of Naive Bayes, Markov chain, and HMM (Hidden Markov Model), and DNN. In order to compare with baseline methods, we tested the methods of Doc2Vec and Word2Vec. Moreover, we investigated the performance of RNN-attention, which is popular in solving time sequence problems, and applied the attention mechanism on RNN.

(1) **MRU** refers to the app that was the most recently used by each user [4];



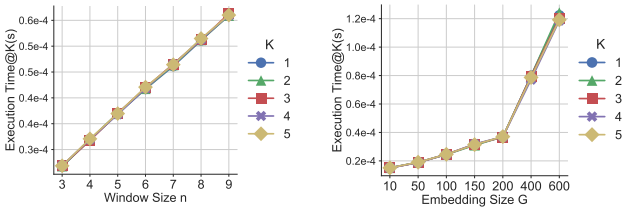
(a) Window size (b) Embedding size

Fig. 4: Prediction performance with varying (a) window size and (b) embedding size.

(2) **MFU** refers to the app that was the most frequently used by each user [4];
(3) **Naive Bayes** takes the apps in a sequence as features and learns their independent contribution to the target app [14];
(4) **Markov chain** estimates the joint probability of app sequence and the target app under the Markov chain rule [13];
(5) **HMM** [19] forms a Markov chain with hidden states that are mapped to the observed app, and the probability distribution of the observed app depends on the hidden states.
(6) **Word2Vec** [23] embeds apps as vectors, and averages the app vectors to predict the next app;
(7) **Doc2Vec** [10] embeds apps and a user as vectors, and concatenates the averaged app vectors with the user vector to predict the next app;
(8) **DNN** [31] stacks fully-connected neural layers to learn nonlinear, hidden and implicit features from an app sequence, and then applies logistic regression with softmax activation for classification.
(9) **RNN-attention** [36] represents app sequences by neurons with recurrent connections, keeps memory about the history in the hidden layers, and then applies attention mechanism to measure the weight of the hidden layer at each time step for the target app. The softmax activation is applied on the weighted summation of the hidden states and connected to a fully-connected layer to estimate the probability distribution of the next app.

B. Results and Analysis

*1) **Performance study w.r.t window size and embedding size.**:* We first investigated the performance of AppUsage2Vec with varying window size n and embedding size G, shown in Fig. 4. Window size n refers to the number of apps input for predicting the next app, and embedding size G refers to the dimensionality of the user vector and app vector. In our experiments, the user and app vectors were represented in the same dimensionality. We also investigated the execution time of AppUsage2Vec with varying window size and embedding size, shown in Fig. 5, to make a tradeoff between the performance and execution time when setting the window size and embedding size. Here, execution time refers to the average running time of each test sample for a specific window size and embedding size, which is calculated by dividing the total running time by the number of all the test samples. The



(a) Window size (b) Embedding size

Fig. 5: Execution time with varying (a) window size and (b) embedding size.

accuracy and execution time were both the average values in 5 runs with the same experiment setting.

As shown in Fig. 4(a), we tested Recall@k (k=1,2,3,4,5), by varying window size from 3 to 9. It can be seen that, Recall@1 increases when window size varied from 3 to 6, suggesting adding recently used apps can bring more historical information for the prediction task to improve the performance. But some results slightly decrease when window size varied from 8 to 9. It is probably because inputting more apps for prediction increase the difficulty in fitting the model. When we used more apps to predict the target apps, there are relatively few sample sequences for training to fit the model. For example, there are much fewer training sample sequences (2,407,331 *vs.* 6,721,261) when inputting 9 apps than that of inputting 3 apps to predict. In addition, inputting more apps makes the computation complexity of the model higher, which also probably makes it difficult to fit the model.

The model achieves the best Recall@1 of 55.45% when the window size is 6. Recall@2, Recall@3, Recall@4, and Recall@5 increase relatively quickly when window size varies from 3 to 4, slowly when the window size varies from 4 to 6. The Recall@2 and Recall@3 are the best when window size is 6, and the Recall@4 and Recall@5 are the best when the window size is 5. It is found that there is a slight difference between recall when the window size varies from 4 to 6. It shows that a longer list of previously used apps does not necessarily provide more useful information than a shorter list in terms of predicting the next app, which coincides with the conclusion in [11].

The execution time of AppUsage2Vec with varying window size is shown in Fig. 5(a). As we can see, the execution time grows linearly with respect to the window size. For each window size, the execution time for $k = 1, 2, 3, 4, 5$ is almost the same. We made a tradeoff between prediction performance and computational efficiency, and set the window size to be 4 in the following experiments. 7,998,707 sampled

sequences were used in total for the experiments, each of which has 5 apps, since we used 4 apps to predict the next app. 5,354,847 sequence samples were used for training, and the left 2,643,860 for the test. For the 8739 users, each user has around 909 sampled sequences in average.

We tested Recall@k with varying the embedding size of the user vector and app vector (10, 50, 100, 150, 200, 400, and 600) when window size was 4, shown in Fig. 4(b). As we can see, Recall@k ($k = 2, 3, 4, 5$) increases relatively quickly when the embedding size varies from 10 to 200, and increases slowly from 200 to 600. Recall@1 increases relatively slowly when the embedding size varies from 10 to 100, and from 200 to 600, while it increases relatively quickly from 100 to 200. Fig. 5(b) shows the execution time of AppUsage2Vec with varying embedding size. It can be seen that the execution time grows relatively slowly when the embedding size varies from 10 to 200, while grows dramatically from 200 to 600. We made a tradeoff between the performance and execution time, and set the embedding size to 200 in the following experiments.

*2) **Comparison with other approaches**:* We compared AppUsage2Vec with other approaches, shown in Tab. I. All of the approaches used four apps to predict the next app. In practice, we tuned different parameters for each approach to achieve the best performance. MRU and MFU were computed based on each sampled sequence. In a sampled sequence of four apps, MRU took the latest used app as the result, and MFU took the most frequently used app of the four apps as the result. If all of the four apps were used once, the result of MFU was the same as that of MRU. We built a Markov chain, an HMM with four hidden states, and developed a DNN with two hidden layers for prediction. RNN-attention applied an attention mechanism to learn the weight of the hidden state at each time step, and a softmax activation was applied on the weighted summation of the hidden states and connected to a fully-connected layer to estimate the probability distribution of the next app. For Word2Vec, Doc2Vec, DNN, and RNN-attention, each input app was represented as a vector of 200 dimensions, and the top k based loss function we proposed in Eq. (16) was applied. We also considered the features of user and time in the models of naive Bayes, Markov chain, HMM, DNN, and RNN-attention. For example, for DNN and RNN-attention, we concatenated the user vector, app vector and time vector together for input. When implementing AppUsage2Vec, we set the discount coefficient α in the top k based loss function to be 0.1, and designed the dual-DNN module with 2 layers.

As shown in Tab. I, AppUsage2vec performs best in Recall@1, Recall@2, Recall@3, Recall@4, and Recall@5. There is a significant improvement of our model, especially in Recall@1. With respect to Recall@1, AppUsage2vec achieves a substantial improvement, making it around 47% higher than naive Bayes, 28% than MRU, 29% than MFU, 22% than Markov chain, 15.5% than Word2Vec, 15% than Doc2Vec, 12% than DNN, 6% than HMM, and 3% than RNN-attention. The difference in Recall@k between AppUsage2Vec and other approaches becomes smaller as k increases. Naive Bayes performs worst, which treats each feature used independently with each other for the target app. The result suggests that the correlation among features used is important for predicting the next app.

TABLE I: Performance comparison with other approaches.

Method	Recall@1	Recall @2	Recall @3	Recall @4	Recall @5
MRU	26.70%	41.15%	56.08%	62.37%	-
MFU	25.50%	45.58%	60.27%	62.37%	-
Naïve Bayes	7.88%	15.61%	22.28%	28.18%	33.54%
Markov chain	32.55%	55.00%	65.62%	72.52%	77.05%
Word2Vec	39.38%	62.81%	72.51%	78.39%	81.66%
Doc2Vec	39.91%	64.54%	73.23%	79.00%	82.76%
DNN	42.86%	66.36%	74.43%	78.79%	83.86%
HMM	48.81%	61.04%	66.26%	72.24%	76.16%
RNN-attention	51.26%	68.99%	76.43%	80.38%	83.36%
AppUsage2Vec	**54.83%**	**69.06%**	**77.63%**	**81.78%**	**84.47%**

HMM achieves a much better Recall@1 (48.81% *v.s.* 32.55%) than Markov chain that assumes the next app is only related to the latest used app. HMM breaks the assumption and introduces hidden states which are the mixture of apps. However, it considers the contribution of the input apps in a given sequence to be the same for the target app, and performs worse than AppUsage2Vec (48.81% *vs.* 54.83%). Our model performs much better in Recall@1 (54.83% *v.s.* 42.86%) than DNN. Although the neural network integrates deep features to encode app usage patterns, DNN makes each input app in a sequence contribute equally for the prediction task. We also found AppUsage2Vec have a significant performance improvement compared to the original Doc2Vec and Word2Vec models, especially in Recall@1 (54.83% *vs.* 39.91% *vs.* 39.38%), suggesting the importance of measuring the contribution of each app, learning user personalized characteristics, and introducing temporal context.

AppUsage2Vec performs around 3.6% higher than RNN-attention in Recall@1, 1.2% higher in Recall@k ($k = 3, 4, 5$). RNN-attention gives very close Recall@2 with AppUsage2Vec, but AppUsage2Vec performs always slightly better than RNN-attention. Although RNN-attention models the sequential correlation among apps and measures the weight of the apps in a given sequence, it is not good at modeling users from their app usage behaviors. Our AppUsage2Vec learns the user personalized characteristics in app usage behaviors by introducing dual-DNN which models the app usage behaviors of each user through the deep neural network, and learns one user's preferences to different apps through Hadamard product on the user and the app sequence vector.

Although the improvements in Recall@k are not so substantial, we found that AppUsage2Vec holds a distinct advantage over RNN-attention in execution time for the prediction. RNN-attention takes approximately twice the execution time of AppUsage2Vec for app usage prediction, as shown in Fig. 6. Here, the execution time refers to the average running time of each test sample where four apps were used to predict the next app. It was calculated by dividing the total running time by the number of all the test samples, and averaged in 5 runs with the same experiment setting. In addition, the

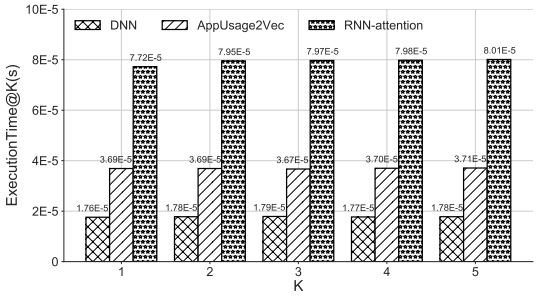


Fig. 6: Execution time comparison among DNN, RNN-attention and AppUsage2Vec.

TABLE II: Effectiveness of different components.

Baseline	Recall@1	39.91%
Baseline+Temporal context	Recall @1	43.39%
	Improvement	**3.48%**
Baseline+Attention	Recall @1	47.36%
	Improvement	**7.45%**
Baseline+dualDNN	Recall @1	44.25%
	Improvement	**4.34%**
Ours	Recall @1	54.83%
	Improvement	**14.92%**

training time of AppUsage2Vec is around 5 minutes shorter than RNN-attention (601.90 *vs.* 908.28 seconds) over all the training samples in the same computation environment. The light-weight implementation, especially the execution time for prediction, is required and very important, since the goal of app usage prediction is to enhance user experience. DNN takes the shortest time for prediction and training (454.49 seconds), but its performance was the worst, 9% lower than RNN-attention and 12% lower than AppUsage2Vec in Recall@1. Thus, AppUsage2Vec, not only with the best performance but also light-weight, is preferred for app usage prediction, which can be allocated to run on a client device.

*3) **Effectiveness of different components**:* We investigated the effectiveness of the components to performance, including temporal context, app-attention mechanism, and dual-DNN. To be specific, we took Doc2Vec as a baseline model, in which the 4 input app vectors were averaged and then concatenated with the user vector. We added temporal context, app-attention mechanism, and dual-DNN to the baseline model, and then compared the Recall@k with the baseline model, respectively. We took Recall@1 to illustrate the effectiveness of the three components shown in Tab. II, since AppUsage2Vec improves the most in Recall@1 when compared to the Doc2Vec model. In Tab. II, 'Ours' means AppUsage2Vec, the model combining together the temporal context, app-attention mechanism and dual-DNN. As we can see from Tab. II:

1) *Temporal context*: Compared to the baseline model, adding temporal context gets Recall@1 3.48% higher. It further validates the importance of time in predicting the next app.
2) *App-attention mechanism*: The app-attention mechanism makes the prediction performance around 7.45% higher in Recall@1. Among the three components, the app-attention mechanism is the most effective one to improve performance. It shows the substantial effect of the app-attention mechanism in app usage prediction. It can capture the users' intention in using apps, and detect which app is important for predicting the next app. The model can still perform well especially in the scenarios when there are some drop-in apps, which are interrupting the app usage patterns.
3) *Dual-DNN*: The component of dual-DNN contributes 4.34% performance gain. The dual-DNN learns user personalized characteristics in app usage. The improvement shows the importance of the user personalized characteristics in predicting next app, including app usage patterns and one's preferences to each app.
4) *The combination of the three components*: Compared to the baseline model, ours improves significantly, 14.92% higher in Recall@1. By combining the three components, our model comprehensively considers the factors of time, the contribution of different apps for the target app, and users' personalized characteristics in app usage.

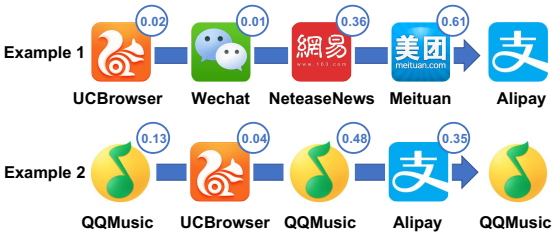
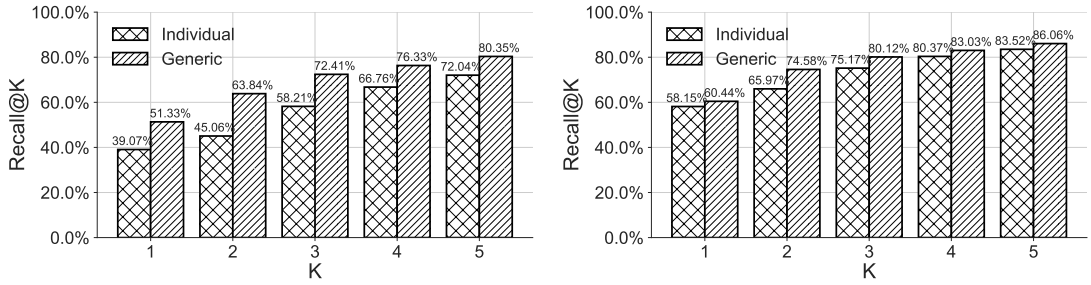


Fig. 7: Two examples of app attention.

*4) **Result illustration of app-attention mechanism**:* In order to better understand the nature of the app-attention mechanism, we inspected two examples from the results, where the attention weight of each app was learned by the app-attention mechanism. In each example, 4 apps were used to predict the next app. In the first example, the user sequentially used the apps of UC browser, WeChat (a popular IM app in China), NeteaseNews, and Meituan (a Chinese Groupon app for locally found consumer products and retail services), which were used for predicting the next app. By the app-attention mechanism, their attention weight is 0.02, 0.01, 0.36, and 0.61, respectively. Meituan is with the highest weight, while WeChat has the lowest weight. It is reasonable, because we checked the usage record and found the next used app is Alipay (an online payment). The user probably used Meituan for online group buying, and then used Alipay for payment. As we expect, the model paid the most attention to Meituan and took it as the most important hint for the target app.

In the second example, QQMusic that was thirdly used has the highest attention weight for the target app. The user likely listened to music via QQMusic (used firstly), then used UCBrowser (used secondly) for searching the song, and got back to QQMusic (used thirdly) to continue to listen to the music. Maybe he/she was interested in the song, he/she went to Alipay (used last) to purchase the song, and then continued to use QQMusic (ground truth) to listen to music. Notably, even for the same app of QQMusic, they have different attention weight, due to different orders in the sequence.



(a) For the users with 200 sequence samples in average

(b) For the users with 4000 sequence samples in average

Fig. 8: Performance comparison between individual model and generic model.

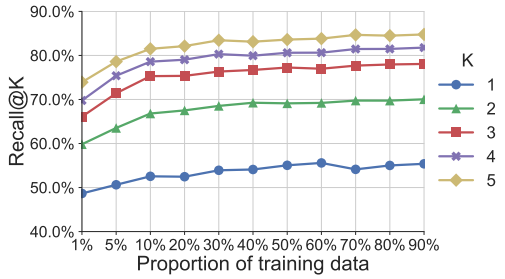


Fig. 9: Performance with partial data for training.

*5) **Performance on different amounts of training data**:* We also investigated the performance of our model trained by different amounts of training data (1%, 5%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, and 90%), shown in Fig. 9. Training data 10%, for example, means we randomly selected 10% of the app sequences from the whole dataset as the training data to predict all users' app usage in the last month from Oct. 23 to Nov. 23. It means, we just varied training data size, but kept the testing data as before. The random selection of the training data was carried out 5 times independently.

As we can see from Fig. 9, AppUsage2Vec still performs well when only 1% and 5% of training dataset are used for training, and achieves Recall@1 of 48.66% and 50.63%, and Recall@5 of 73.22% and 79.78%, respectively. The performance increases quickly when from 1% to 10% training data were used. There is a slight difference in performance when from 10% to 90% training data was used. It can be inferred that our model performs stably with varying amounts of training data, even for small training data.

*6) **Generic model to address cold-start problem**:* This experiment was designed to show the effectiveness of our model for solving the cold-start problem. The cold-start problem can be characterized by a prediction model that has difficulty in accurately predicting due to a lack of a sufficient amount of data. Most of the existing models in previous studies, such as MRU, MFU, Naive Bayes and Markov chain, were trained using an individual's data, *i.e.* individual models. However, training an individual model often encounters the cold-start problem, especially at the beginning of data collection for an individual. A generic model that is trained using all available users' data, can provide potentially better prediction performance when the data is not sufficient. AppUsage2Vec can be trained as not only an individual model, but also a generic model. We examined how the generic model can address the cold-start problem for individual users.

We trained a generic model and individual models, and compared the performance of them. To be specific, the generic model was trained using the data from Aug. 23 to Oct. 23 of all the 8,739 users. Individual models were trained using very small data of each user that has relatively few sequences in the three months. We randomly selected 100 users who have around 200 sampled sequences in the three months. Each sampled sequence has 5 apps, since we used 4 apps to predict the next app. The 100 users have 196.07 sampled sequences in average (in the whole dataset, each user has around 909 sampled sequences in average). Each individual model was trained using each individual's data from Aug. 23 to Oct. 23, and there were 100 individual models in total. Both individual models and the generic model were tested using the 200 users' data in the last month from Oct. 23 to Nov. 23, 2017. For the generic and individual models, Recall@k was the average value of the 100 users, shown in Fig. 8(a). For comparison, we selected the top 100 users who has the most sequence samples (3951.96 in average), trained their individual models in the same way, and compared the performance between the individual model and generic model, shown in Fig. 8(b).

As shown in Fig. 8(a), the generic model performs much better against individual models. To be specific, for Recall@1, Recall@2, Recall@3, Recall@4, and Recall@5, generic model performed around 12%, 18%, 14%, 10%, and 8% higher than individual models, respectively. This suggested that the generic model trained using all available users' data models can significantly help address the cold-start problem for individual users and enhance performance, especially at the beginning of data collection. As we expect, the performance of the individual model increases when much more training data is used, shown in Fig. 8(b). But, the generic model still outperforms the individual models, indicating the superiority of the generic model. The differences between the individual and generic model greatly increase as the training data size becomes smaller. It further suggests the generic model can help solve the cold-start problem when there is not sufficient data to train individual models.

VI. Conclusions

In this work, we have predicted the next app most likely to be used based on the most recently used apps, by proposing a novel framework, AppUsage2Vec, inspired by Doc2Vec. Com-

pared to Doc2Vec, we introduced an app-attention mechanism to measure the contribution of each app to the target app, designed a module of dual-DNN to learn user personalized characteristics, and took temporal context into consideration. Furthermore, we proposed a top-k based loss function for training. Extensive experiments were conducted on a real-world dataset of 10,360 users in three months for evaluation. We investigated the performance of our model when it was an individual model. The generic model performed much better than individual models, especially when each individual's training data was very small, which can provide potentially better performance to solve the cold-start problem. The results showed that AppUsage2Vec is light-weight for app usage prediction, which can be allocated to run on a client device. The results demonstrate the state-of-the-art performance, achieving recall of 54.83% and 84.47% for top-1 and top-5, respectively.

AppUsage2Vec can not only predict the next app that most likely to be used next, but also learn user and app representations. The user and app representations are interesting because the learned vectors explicitly encode many app usage patterns. They might be extended for other application scenarios, for example, app categorization, and discovery of user groups with similar usage patterns. It could be used to facilitate the efficient marketing of products and services.

ACKNOWLEDGMENT

This work was supported by NSF of China (No. 61802342, 61772460, and 61802340), China Postdoctoral Science Foundation (No. 2017M620246 and 2018T110591), and Fundamental Research Funds for the Central Universities. Dr. Shijian Li and Dr. Gang Pan are the corresponding authors.

REFERENCES

[1] Z.-X. Liao, S.-C. Li, W.-C. Peng, S. Y. Philip, and T.-C. Liu, "On the feature discovery for app usage prediction in smartphones," in *ICDM 2013*. IEEE, 2013, pp. 1127–1132.

[2] H. Cao and M. Lin, "Mining smartphone data for app usage prediction and recommendations: A survey," *Pervasive and Mobile Computing*, pp. 1–22, 2017.

[3] Z.-X. Liao, Y.-C. Pan, W.-C. Peng, and P.-R. Lei, "On mining mobile apps usage behavior for predicting apps usage in smartphones," in *CIKM 2013*. ACM, 2013, pp. 609–618.

[4] C. Shin, J.-H. Hong, and A. K. Dey, "Understanding and prediction of mobile application usage for smart phones," in *UbiComp 2012*. ACM, 2012, pp. 173–182.

[5] N. Natarajan, D. Shin, and I. S. Dhillon, "Which app will you use next?: collaborative filtering with interactional context," in *RecSys 2013*. ACM, 2013, pp. 201–208.

[6] K. Huang, C. Zhang, X. Ma, and G. Chen, "Predicting mobile application usage using contextual information," in *UbiComp 2012*. ACM, 2012, pp. 1059–1065.

[7] R. Baeza-Yates, D. Jiang, F. Silvestri, and B. Harrison, "Predicting the next app that you are going to use," in *WSDM 2015*. ACM, pp. 285–294.

[8] D. Yu, Y. Li, F. Xu, P. Zhang, and V. Kostakos, "Smartphone app usage prediction using points of interest," *IMWUT*, vol. 1, no. 4, p. 174, 2018.

[9] S. Zhao, G. Pan, Y. Zhao, J. Tao, J. Chen, S. Li, and Z. Wu, "Mining user attributes using large-scale app lists of smartphones." *IEEE Systems Journal*, vol. 11, no. 1, pp. 315–323, 2017.

[10] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *ICML2014*, 2014, pp. 1188–1196.

[11] A. Parate, M. Böhmer, D. Chu, D. Ganesan, and B. M. Marlin, "Practical prediction and prefetch for faster access to applications on mobile phones," in *UbiComp 2013*. ACM, 2013, pp. 275–284.

[12] C. Sun, J. Zheng, H. Yao, Y. Wang, and D. F. Hsu, "Apprush: Using dynamic shortcuts to facilitate application launching on mobile devices," *Procedia Computer Science*, vol. 19, pp. 445–452, 2013.

[13] T. M. T. Do and D. Gatica-Perez, "Where and what: Using smartphones to predict next locations and applications in daily life," *Pervasive and Mobile Computing*, vol. 12, pp. 79–91, 2014.

[14] X. Zou, W. Zhang, S. Li, and G. Pan, "Prophet: What app you wish to use next," in *UbiComp 2013 poster*. ACM, 2013, pp. 167–170.

[15] Y. Xu, M. Lin, H. Lu, G. Cardone, N. Lane, Z. Chen, A. Campbell, and T. Choudhury, "Preference, context and communities: a multi-faceted approach to predicting smartphone app usage patterns," in *ISWC 2013*. ACM, 2013, pp. 69–76.

[16] C. Xiang, D. Liu, S. Li, X. Zhu, Y. Li, J. Ren, and L. Liang, "Hinextapp: A context-aware and adaptive framework for app prediction in mobile systems," in *2017 IEEE Trustcom/BigDataSE/ICESS*, pp. 776–783.

[17] H. Yin, L. Chen, W. Wang, X. Du, Q. V. H. Nguyen, and X. Zhou, "Mobi-sage: A sparse additive generative model for mobile app recommendation," in *ICDE 2017*. IEEE, 2017, pp. 75–78.

[18] Y. Wang, N. J. Yuan, Y. Sun, F. Zhang, X. Xie, Q. Liu, and E. Chen, "A contextual collaborative approach for app usage forecasting," in *UbiComp 2016*. ACM, 2016, pp. 1247–1258.

[19] B. Huai, E. Chen, H. Zhu, H. Xiong, T. Bao, Q. Liu, and J. Tian, "Toward personalized context recognition for mobile users: A semisupervised bayesian hmm approach," *TKDD*, vol. 9, no. 2, p. 10, 2014.

[20] H. Zhu, E. Chen, H. Xiong, K. Yu, H. Cao, and J. Tian, "Mining mobile user preferences for personalized context-aware recommendation," *TIST*, vol. 5, no. 4, p. 58, 2015.

[21] H. Cao, T. Bao, Q. Yang, E. Chen, and J. Tian, "An effective approach for mining mobile user habits," in *CIKM 2010*. ACM, pp. 1677–1680.

[22] K. Yu, B. Zhang, H. Zhu, H. Cao, and J. Tian, "Towards personalized context-aware recommendation by mining context logs through topic models," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2012, pp. 431–443.

[23] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems*, 2013, pp. 3111–3119.

[24] Q. Ma, S. Muthukrishnan, and W. Simpson, "App2vec: modeling of mobile apps and applications," in *ASONAM 2016*. IEEE, pp. 599–606.

[25] V. Radosavljevic, M. Grbovic, N. Djuric, N. Bhamidipati, D. Zhang, J. Wang, J. Dang, H. Huang, A. Nagarajan, and P. Chen, "Smartphone app categorization for interest targeting in advertising marketplace," in *WWW 2016*, 2016, pp. 93–94.

[26] C. Jesdabodi and W. Maalej, "Understanding usage states on mobile devices," in *UbiComp 2015*. ACM, 2015, pp. 1221–1225.

[27] Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman, "Identifying diverse usage behaviors of smartphone apps," in *IMC 2011*. ACM, 2011, pp. 329–344.

[28] S. Zhao, J. Ramos, J. Tao, Z. Jiang, S. Li, Z. Wu, G. Pan, and A. K. Dey, "Who are the smartphone users?: Identifying user groups with apps usage behaviors," *GetMobile: Mobile Computing and Communications*, vol. 21, no. 2, pp. 31–34, 2017.

[29] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[30] S. Zhao, F. Xu, Z. Luo, S. Li, and G. Pan, "Demographic attributes prediction through app usage behaviors on smartphones," in *UbiComp 2018 workshop*. ACM, pp. 870–877.

[31] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[32] S. Zhao, J. Ramos, J. Tao, Z. Jiang, S. Li, Z. Wu, G. Pan, and A. K. Dey, "Discovering different kinds of smartphone users through their application usage behaviors," in *UbiComp 2016*. ACM, pp. 498–509.

[33] M. Lapin, M. Hein, and B. Schiele, "Top-k multiclass svm," in *Advances in Neural Information Processing Systems*, 2015, pp. 325–333.

[34] H. Yao, G. Ranjan, A. Tongaonkar, Y. Liao, and Z. M. Mao, "Samples: Self adaptive mining of persistent lexical snippets for classifying mobile application traffic," in *MobiCom 2015*. ACM, 2015, pp. 439–451.

[35] H. Cao, D. H. Hu, D. Shen, D. Jiang, J.-T. Sun, E. Chen, and Q. Yang, "Context-aware query classification," in *SIGIR 2009*. ACM, pp. 3–10.

[36] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Eleventh Annual Conference of the International Speech Communication Association*, 2010.