Electronic Preprint for Journal of Information Processing Vol.29

Regular Paper

Predicting Next-use Mobile Apps Using App Semantic Representations

Cheng Chen[1,a) Takuya Maekawa[1,b) Daichi Amagata[1,c) Takahiro Hara[1,d)

Received: December 15, 2020, Accepted: July 7, 2021

Abstract: Using the app usage history of a target user as a basis, this study proposes a novel method for predicting next-use mobile apps of the user that can assist the user in selecting an app from a list of installed apps. The proposed method is designed to train a next-use app prediction model using semantic representations of the usage histories of other users (source users) to deal with the user and app cold-start problems of an app prediction system in which training data from a target user beginning to use the system and training data related to newly installed or released apps are considered to be insufficient. We predict the usage of apps by a target user by leveraging the semantic similarities between the apps that are installed on the smartphones of the source users and the apps that are installed on the smartphone of a target user, permitting us to predict next-use apps regardless of the apps installed in the target user's smartphone. We evaluate our method using the actual app usage data collected from 100 participants over a period of approximately 70 days with 300,000 app usage histories.

Keywords: smartphone, mobile apps, next-use app prediction, pattern recognition

1. Introduction

1.1 Background

As a result of the recent proliferation of smartphones, the number of available smartphone apps in app stores is rapidly increasing. This huge number and diversity of apps enables the installation of a large number of apps on a user's smartphone. In fact, Bazea-Yates et al. [1] revealed that 96 apps are installed on a smartphone on average. Although the large number of available apps makes our lives more convenient, it also introduces a new challenge in that selecting a particular app from the already installed apps can be time-consuming. To assist a user in selecting apps efficiently, methods for predicting a next-use app have been actively studied in ubiquitous computing, mobile computing, and recommender system research [2], [3], [4]. These methods can help the user find apps by providing probable candidates when the user attempts to show a list of installed app icons. Mobile apps are often used in conjunction with other relevant apps [5]. For instance, when a user uses a smartphone for his or her business, he or she might initially use a word processor app. When the user wants to send the edited document to others, the next-use app is likely to be an e-mail app. Considering this fact, a next-use app can be predicted on the basis of a user's app usage history. Some previous studies have used supervised learning methods to generate predictions based on app usage histories [6], [7].

1.2 Problems

Supervised learning-based next-use app prediction poses the following problems.

- **User cold-start problem**: The prediction model cannot be trained immediately after a user installs the next-use app-prediction system because such a user has no usage history.
- **App cold-start problem**: The prediction model cannot recommend an app immediately after a user installs a new app from an app store because the user has no usage history with this new app.

To alleviate these cold-start problems, some studies have considered leveraging other users' (source users) usage histories to construct a prediction model for a target user [1]. However, using the usage history of source users also poses various problems, as follows.

- Apps that are installed on the smartphones of source users are different from those installed on the smartphones of the target user, making it difficult to recommend to the target user apps that are not installed on the smartphones of source users (especially for newly released apps).
- Apps that are pre-installed by smartphone manufacturers, such as the setting and camera apps, are difficult to recommend because pre-installed apps are different for different types of smartphones.
- Apps that are directly installed from Android application package (APK) files, rather than obtained from an app store, are difficult to recommend because few users use them.

In our dataset, 16% of the installed apps were categorized as unseen apps belonging to one of these three types.

[1] Graduate School of Information Science and Technology, Osaka University, Suita, Osaka 565–0871, Japan
[a) chch9278@gmail.com
[b) maekawa@ist.osaka-u.ac.jp
[c) amagata.daichi@ist.osaka-u.ac.jp
[d) hara@ist.osaka-u.ac.jp

1.3 Approach

To deal with these problems, we propose a next-use app prediction method based on semantic representations of a smartphone app. Specifically, for apps that installed from app store, we leverage a description of an app to build an app vector that serves as a semantic representation of the app. For system apps (e.g., camera app) and apps directly installed from Android application package (APK) file, we used the title of an app to build an app vector that serves as a semantic representation of the app. Let us assume that a target user has installed a newly released puzzle game app. Even when the usage history for this new app is unavailable in the usage history of the source users, our approach can recommend this puzzle app to the target user if the usage of semantically similar puzzle games is contained in the source users' usage history, because it can be assumed that the already installed puzzle games and the newly released puzzle game exhibit similar usage patterns. Consequently, we propose introducing semantic representations of apps, in the form of app vectors, into a prediction model. Because an app vector permits us to build a model that does not depend on apps that are installed on each user's smartphone, apps that are not installed on the source users' smartphones (unseen apps) can be recommended to the target user. To capture usage patterns of apps effectively, the proposed method has the following features.

(1) We design the app prediction method based on app vectors to address the uncertainty of a target user's next-use apps. A simple prediction method based on app vectors is assumed to output a vector showing an estimate of the user's next-use app using a regression technique and then to recommend apps having vectors similar to the estimated vector. This results in low diversity in a ranked list of recommended apps, because only apps similar to the estimated vector are included in the list. Our method is designed to output a diversity of apps by introducing vector representations of apps into a multi-class classifier that is expected to output diverse results in a ranked list. Our idea is to convert time series of app usages by source users into time series of usages of apps installed in the target user's smartphone represented in the 1-of-K scheme [8] using semantic similarities between apps, thereby permitting us to train a multi-class classifier tailored to the target user that can output a ranked list containing diverse apps. This is explained in detail later.

(2) Because usages of apps by a user are considered to comprise a time series, we design our method on the basis of long short-term memory (LSTM) [9] neural networks. However, our investigation revealed that app usage patterns can be strongly related to the time intervals of app usages, a feature impossible to handle with LSTM networks. To deal with this fact, we propose interval-LSTM (*i*-LSTM) networks by modifying gates in a traditional LSTM cell to capture the time intervals of input data.

(3) Because a trained *i*-LSTM network on the source users can be regarded as a user-independent model and does not always capture the usage pattern of a target user, we periodically fine-tune the trained network by leveraging the target user's accumulated usage history.

1.4 Contributions

The contributions of this study are as follows.

- We use other users' (source users) usage histories to alleviate the user and app cold-start problems in next-use app prediction.
- We designed a next-use app prediction architecture based on deep learning and app semantic representations that can predict the usage pattern of an app that is not installed on the source users' smartphones.
- We evaluate our method using an actual app usage dataset collected from 100 participants. Surprisingly, for unseen apps (apps installed on a target user's smartphone but not installed on source users' smartphones), our method achieves 62% accuracy in the top-N recommendation tasks ($N = 10$). It is important to note that conventional approaches cannot predict the usage of such unseen apps.

2. Related Work

2.1 Next-use App Prediction

The next-use app prediction task involves predicting the i^{th} app used by a user when the user's app usage history through $i-1$ is provided. In many previous studies, the next-use app prediction problem was formulated as a multi-class classification problem. We introduce some previous studies related to next-use app prediction. Zou et al. [7] proposed some light-weighted Bayesian methods to predict a next-use app. Sun et al. [6] used a prediction model that utilized app temporal features such as frequency and duration. Liao et al. [10] designed a time-based app predictor, extracting some features from the app usage trace, such as an app's usage count in the entire usage trace, the usage count in the temporal bucket, and the frequency of app usage. In contrast, we attempt to deal with the cold-start problems in next-use app prediction. Further, our method, based on deep learning, does not require handcrafted features.

Some other studies used sensor data, such as data from the global positioning system (GPS), for generating predictions. Shin et al. [11] proposed a context model for app prediction that used an extensive variety of contextual information from sensors in a smartphone and constructed a personalized app-prediction model based on naive Bayes. Huang et al. [12] explored various types of contextual information, such as the last used app, time, location, and user profile, to predict the user's app usage [13]. Liao et al. [14] proposed an app usage prediction framework that uses both explicit data from mobile sensors and implicit transitions across app usage. Bazea-Yates et al. [1] collected multiple sensor data from a home screen app Aviate and built a parallel tree-augmented naive Bayes model to generate predictions. Wang et al. [15] proposed a long-term app usage forecasting method based on the contextual information related to location.

2.2 Cold-start Problems in Next-use App Prediction

As mentioned in the introduction, cold-start problems in this research are divided into two types, the user cold-start problem, in which a first-time user used the app recommendation system, and the app cold-start problem, in which a user installed a new app one his or her smartphone. To solve the user cold-start problem, Xu et al. [16] used a user community pattern and learned a target user's app usage pattern from similar users when the training data of the target user was considered to be insufficient. Bazea-Yates et al. [1] used the app usage information obtained from other users for generating predictions to alleviate the user cold-start problem. They used a similar user's model to predict the behavior of the new user. They also compared the installed apps of new and other users to determine similar users. For the app cold-start problem, Lin et al. [17] proposed a method that accounted for nascent information obtained from Twitter to provide relevant recommendations. Natarajan et al. [5] investigated the both app and user cold-start problems. For the app cold-start problem, they assumed that a user is more likely to prefer an item belonging to the same genre than an item belonging to a different genre after using a series of the same type of items. In accordance with that assumption, they recommended to a user a new app belonging to the same genre as that of the previously used apps. For the user cold-start problem, they created a new user's usage history based on a uniform distribution over all the apps. In contrast to the aforementioned studies, we use high-level app semantic information to alleviate the cold-start problem, enabling us to predict the usage patterns of apps that are not installed on the smartphones of source users.

2.3 Problems in Next-use App Prediction Architectures in Existing Studies

As mentioned above, the next-use app prediction problem was formulated as a multi-class classification problem in many previous studies. When K apps are installed on a user's smartphone, each app is represented in terms of the 1-of-K scheme [8] in which an app is represented by a K-dimensional vector, where one of the elements is one and the remaining elements are zero. For instance, a K-dimensional vector [0 0 1 0 0 0 ...] represents the usage of the third app. A label of the training data for the prediction model is also represented using the 1-of-K scheme. Consequently, an output of the prediction model is a K dimensional vector. An example of an estimate of the trained prediction model, i.e., the next-use probability of each app, is [0.0 0.1 0.3 0.5 ...], which indicates that the probability for the second app is 0.1, the probability for the third app is 0.3, and so on. Based on such an estimate, the top-N apps in terms of the next-use probabilities are recommended to the user. The prediction model (multi-class classifier) estimates the next-use probability for each independent app, diversifying the top-N apps. However, methods based on the 1-of-K scheme do not work when apps that are installed on the smartphones of a target user are not available on the smartphones of source users, because the 1-of-K representation includes only information regarding the identifiers of apps.

3. Problem Formulation

We use source users' usage histories to train a prediction model and then generate predictions for a target user. We define an app, source users' U, and target user's u_t as follows:

DEFINITION 1 (APP).
A set of apps installed on a user's smartphone, viz., $\mathcal{A} = \{a_{\grave{1}}, a_{\grave{2}}, a_{\grave{3}}, \ldots, a_{\grave{n}}, \ldots, a_{\grave{K}}\}$, where $a_{\grave{n}}$ is the n^{th} app in the set. In contrast, the i^{th} used app in a user's usage history can be represented as a_i. When $a_{\grave{n}}$ is the i^{th} used app, a_i becomes equal to $a_{\grave{n}}$. A user's i^{th} used app a_i is represented as a semantic vector v_{a_i}. The app is also represented as $o^K_{a_i}$ in accordance with the 1-of-K scheme, where K is the number of dimensions of the vector, i.e., the number of apps installed on the user's smartphone. In addition, we refer to an app that is not installed on the source users' smartphones but is installed on the target user's smartphone as an unseen app. Furthermore, we refer to an app that is installed on both the source users' smartphones and the target user's smartphone as an existing app.

DEFINITION 2 (SOURCE USERS).
A set of N users $U = \{u_1, u_2, u_3, \ldots, u_N\}$. Each user u_i $(1 \leq i \leq N)$ has an app usage history with length M_i, viz., $\mathcal{S}_i = \{a_1, a_2, a_3, \ldots, a_{M_i}\}$.

DEFINITION 3 (TARGET USER).
A user $u_t \notin U$ with usage history of length M_t, viz., $\mathcal{S}_t = \{a_1, a_2, a_3, \ldots, a_{M_t}\}$. When we wish to predict a next-use app, we do not use all of the usage histories to generate predictions, because an app that was used in the distant past is likely to exhibit little relation to the latent next-use app. In this study, we use a sequence of app usage histories of lengths k $(1 \leq k \leq M)$, i.e., $s = \{a_{i-k}, a_{i-k+1}, \ldots, a_{i-2}, a_{i-1}\}$, to predict a next-use app a_i. Further, the next-use app prediction problem can be defined as follows:

DEFINITION 4 (APP PREDICTION).
Given a series of app usage histories of a target user u_t with length k $(1 \leq k \leq M)$, i.e., $s = \{a_{u_t,i-k}, a_{u_t,i-k+1}, \ldots, a_{u_t,i-2}, a_{u_t,i-1}\}$, the probability that each app $a_{\grave{n}}$ that is installed on this user's smartphone will be a next-use app, i.e., $P(a_{\grave{n}}|s)$, is calculated. We further select the top-N apps to be the next-use app candidates.

4. Next-use App Prediction Method

4.1 Overview

An overview of our proposed method is shown in **Fig. 1**. The proposed method has two phases, training and prediction. In the training phase, we initially compute an app semantic representation, i.e., an app vector, for each app installed on the source users' or target user's smartphones. Further, we utilize the sequences of usages of apps from the source users' usage histories to generate training data that are tailored to the target user by leveraging the app semantics. That is, we convert the usage of a source user's app in the usage history into the usage of a target user's app that is semantically similar to the source user's app. Subsequently, we train a prediction model for the target user based on a deep neural network consisting of our modified LSTM on the tailored training data. In the prediction phase, we use the user's app usage history through $i - 1$ to predict the target user's i^{th} app usage candidates. In addition, we periodically update the prediction model using the accumulated usage history of the target user.

4.2 Semantic Representation of App

We assume that we obtain a description of each app from the app store, which is used to build a semantic app vector. An app description describes the features of an app and the functions

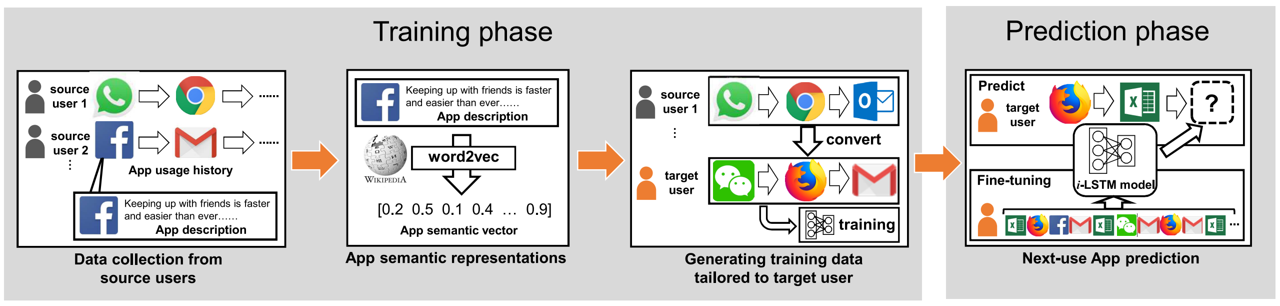

Fig. 1 Overview of the proposed method.

that are to be provided to users. For instance, a description of the Facebook app in Google Play is “Keeping up with friends is faster and easier than ever. Share updates and photos, engage with friends and Pages, and stay connected to communities important to you,” describing the characteristics of the Facebook app. In contrast, a description of the Instagram app in Google Play is “Instagram is a simple way to capture and share the world’s moments. Follow your friends and family to see what they’re up to, and discover accounts from all over the world that are sharing things you love.” As can be observed from these examples, the social network service (SNS) apps exhibit semantically similar descriptions. Therefore, we construct semantic app vectors from descriptions to calculate the similarities between various apps. Note that we cannot obtain the descriptions of apps that are pre-installed or directly installed from APK files. The titles of such apps are used in place of descriptions. In our proposed method, we initially perform the morphological analysis of Japanese descriptions based on Ref. [18], because almost all of the apps used in our experiment are Japanese apps. Therefore, we tokenize a Japanese description, i.e., text segmentation, to extract words from it. Further, we select W keywords of an app that represent the app well from the extracted words, using the importance of the words computed based on $tf\text{-}idf$. The importance of word w is computed using $tf\text{-}idf$, as follows:

$$tf\text{-}idf(w) = Frequency(w) \cdot \log \frac{N}{|d : d \ni w| + 1},$$

where $Frequency(w)$ is the word occurrence frequency of w in the description, and N is the total number of descriptions of all apps. Further, we use a word embedding model word2vec [19] to obtain a word vector for each keyword representing the semantics of the keyword. We use an external pre-trained word2vec model on Japanese Wikipedia [*1] to compute a word vector for each keyword. Finally, we compute the mean vector of the W keywords and regard it as the app’s semantic vector. We ignore keywords that are not present in the external Japanese Wikipedia corpus. We use this method to build an app semantic vector for each app on the source users’ or target user’s smartphones. **Figure 2** depicts the semantic vectors of approximately 6,000 apps constructed using the aforementioned method and projected into two-dimensional space using t-SNE [20]. Each vector is colored according to the category that it belongs to on Google Play. We can observe that approximately all of the apps belonging to the same categories are grouped and that the distance between apps

[*1] http://www.cl.ecei.tohoku.ac.jp/~m-suzuki/jawiki_vector/

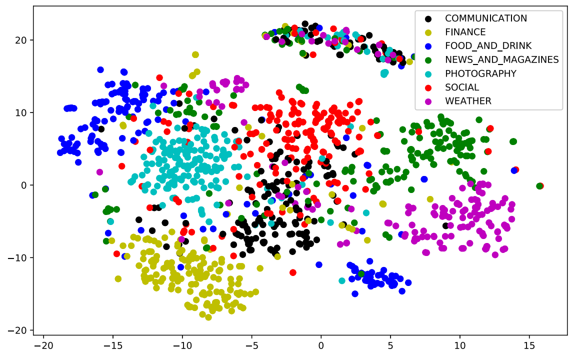

Fig. 2 Visualization of the app semantic vectors.

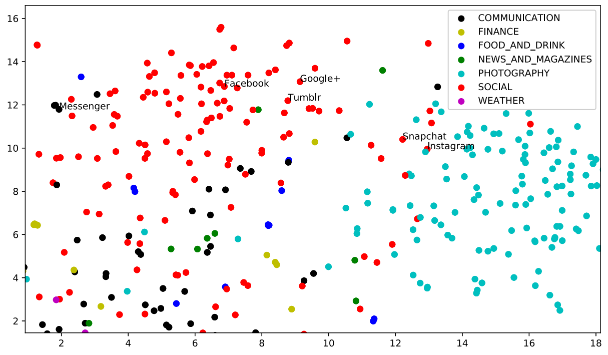

Fig. 3 Visualization of some of the apps around the Facebook app.

belonging to different categories is larger than that for those in the same category. **Figure 3** depicts some of the apps around the Facebook app. We can observe that several apps around the Facebook app, such as Tumblr and Google+, belong to the social category. In addition, Instagram, which belongs to the photography category while having social functionalities, is located on the boundary between the social and photography categories, indicating that the vector representations can capture the fine-grained semantics of apps.

Note that our method does not use information about categories of apps on Google Play. This is because all apps do not have category information and our preliminary investigation revealed that using the category information as additional inputs does not improve the next-use app prediction accuracy. This may be because semantic vectors of apps have already included information similar to the category information as shown in Fig. 2.

4.3 Generating Training Data Tailored to a Target User

To deal with the problem of existing studies not being able to handle unseen apps, we combine multi-class classification based on 1-of-K representations and app semantic vectors. Our basic idea is to generate training data in the 1-of-K representations, which can be used for multi-class classification, tailored to a target user from the source users' usage histories by using app semantic representations. Let us assume that a series of app usages obtained from a source user is provided. We obtain a series of K-dimensional vectors in the 1-of-K representation tailored to the target user from the series of app usages, where K is the number of apps that are installed on a target user's smartphone. Here, because usages of apps that are installed only on source users' smartphones are not directly used for predicting app usages by the target user, the usages of these apps are converted into usages of apps installed on the target user's smartphone. We calculate the semantic similarity between each pair of an app installed on a target user's smartphone and an app installed only on source users' smartphones based on their semantic representations and subsequently convert the app usage history of the source users into a series of 1-of-K representations of a target user's app based on the calculated similarities, i.e., the usage of a source user's app is converted into the usage of a target user's app that is semantically similar to the source user's app. Usages of apps that are installed on both the target and source users' smartphones are not converted. The converted series of K-dimensional vectors is used further to train a multi-class classifier (next-use app prediction model) for the target user.

The procedure of generating training data tailored to a target user is summarized in Algorithm 1, which is applied to each usage sequence of each source user. In essence, we generate probabilistically a variety of sequences of app usages tailored to the target user from a sequence of usages by a source user to achieve robust next-use app prediction. An input of the algorithm is a sequence with length M of a source user's app usages, and an output is sequences of app usages with length M tailored to a target user. After an app usage history with length M tailored to the target user is obtained, we extract $M - (k + 1)$ sequences of app usages with length $k + 1$ from the history and subsequently train the next-use app predictor on the sequences. Initially, we prepare the conversion matrices that convert the usage of a source user's app into the usage of an app installed on the target user's smartphone (lines 3–9). The detail of the conversion matrices generation method is explained in Section 4.3.1. Using the matrices, we probabilistically generate multiple sequences of app usages tailored to the target user from the sequence of source user app usages (lines 11–28). The detail of the app usage sequence generation method is explained in Section 4.3.3. To generate each output sequence, we probabilistically construct a conversion table that describes a mapping from a source user's app to a target user's app (lines 13–24). The detail of the conversion table generation method is described in Section 4.3.2. Because a mapping from a source user's app to a target user's app computed from the app semantics is not entirely accurate, we probabilistically generate a variety of multiple sequences in accordance with the similarities between apps to train a robust next-app predictor. We explain the procedures in detail.

Algorithm 1: Generating training data tailored to a target user

```
Input: $\mathcal{T}$: a set of apps from a target user, $\mathcal{S}$: a sequence of app
       usages from a source user, $\theta$: a parameter used to control
       the quantity of unseen apps in training data
1  $\mathcal{D} \leftarrow \emptyset$
2  /* Extract a set of apps of source user */
3  $\mathcal{A}_s \leftarrow$ ExtractAppSet($\mathcal{S}$)
4  /* Divide target user apps into 2 groups */
5  $\mathcal{G}_e \leftarrow \mathcal{T} \cap \mathcal{A}_s$ /* A set of existing apps */
6  $\mathcal{G}_u \leftarrow \mathcal{T} \cap \bar{\mathcal{G}}_e$ /* A set of unseen apps */
7  /* Creating app conversion matrices */
8  $\boldsymbol{M}_e \leftarrow$ GenerateConversionMatrix($\mathcal{T}, \mathcal{A}_s, \mathcal{G}_e$)
9  $\boldsymbol{M}_u \leftarrow$ GenerateConversionMatrix($\mathcal{T}, \mathcal{A}_s, \mathcal{G}_u$)
10 /* Generating $P$ sequences of training data from $\mathcal{S}$
    */
11 repeat
12     /* Initialize app conversion table */
13     $T \leftarrow \emptyset$
14     for $\forall a_s \in \mathcal{A}_s$ do
15         $\boldsymbol{M} \leftarrow$ zero matrix
16         $x \leftarrow$ Uniform$(0, 1)$ /* Randomly generate $x \in [0, 1]$
            */
17         if $x < \theta$ then
18             $\boldsymbol{M} \leftarrow \boldsymbol{M}_u$
19         else
20             $\boldsymbol{M} \leftarrow \boldsymbol{M}_e$
21         /* Generate mapping from a source app to a
            target app */
22         $a_t \leftarrow$ ConvertApp($\boldsymbol{M}, a_s$)
23         /* Add mapping from a source app to a target
            app */
24         $T[a_s] \leftarrow a_t$
25     /* Convert $\mathcal{S}$ using conversion table $T$*/
26     $\mathcal{D}_p \leftarrow$ ConvertSequence($\mathcal{S}, T$)
27     $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathcal{D}_p\}$
28 until repeat the processing P times
Output: $\mathcal{D}$: a set of training sequences tailored to a target user
```

4.3.1 Generating Conversion Matrices

In this study, we generate two conversion matrices used to control the number of usages of unseen apps that are included in the training data to be generated, i.e., sequences of app usages tailored to the target user, because the number of usages of unseen apps to be included in the training data can be typically less than that of the existing apps that have already been included in sequences of source user app usages, degrading the prediction performance related to the usages of unseen apps. More specifically, we generate a conversion matrix $\boldsymbol{M}_u$ that converts usages of an app installed only on source users' smartphones into usages of unseen apps and another conversion matrix $\boldsymbol{M}_e$ that converts usages of an app that is installed only on source users' smartphones into usages of existing apps. When we generate a sequence of app usages tailored to the target user from a sequence of app usages by a source user, we can generate a sequence containing many usages of unseen apps by using $\boldsymbol{M}_u$ many times. A conversion matrix is used to obtain the candidates for target user apps converted from a source user app as follows.

$$\hat{\boldsymbol{p}}^{|\mathcal{T}|} = \boldsymbol{M}\boldsymbol{o}_{a_s}^{|\mathcal{A}_s|}, \tag{1}$$

where $\boldsymbol{M}$ is the conversion matrix, $\boldsymbol{o}_{a_s}^{|\mathcal{A}_s|}$ is a $|\mathcal{A}_s|$-dimensional vector of a source user's app a_s represented using the 1-of-K scheme, $\mathcal{A}_s$ is a set of apps installed on a source user's smartphone, and $\mathcal{T}$ is a set of apps installed on a target user's smartphone. In addition, $\hat{\boldsymbol{p}}^{|\mathcal{T}|}$ is a $|\mathcal{T}|$-dimensional vector that represents the probability of converting each respective target user app from a_s. Here, we explain the procedures for generating the conversion matrices. After obtaining a set of apps installed on a source user's smartphone $\mathcal{A}_s$ (line 3), we obtain a set of existing apps $\mathcal{G}_e$ (line 5) and a set of unseen apps $\mathcal{G}_u$ (line 6). Using $\mathcal{G}_e$, we generate a conversion matrix for the existing apps $\boldsymbol{M}_e$ (line 8). $\boldsymbol{M}_e$ is used to convert a source user app not installed on the target user's smartphone into an existing app. Furthermore, we generate a conversion matrix for unseen apps $\boldsymbol{M}_u$ using $\mathcal{G}_u$ (line 9). $\boldsymbol{M}_u$ is used to convert a source user's app that is not installed on the target user's smartphone into an unseen app. A conversion matrix $\boldsymbol{M}$, a $|\mathcal{A}_s| \times |\mathcal{T}|$ matrix, can be described as follows.

$$\boldsymbol{M} = \begin{bmatrix} m_{11} & m_{12} & \dots & m_{1|\mathcal{A}_s|} \\ m_{21} & m_{22} & \dots & m_{2|\mathcal{A}_s|} \\ \vdots & \vdots & \ddots & \vdots \\ m_{|\mathcal{T}|1} & m_{|\mathcal{T}|2} & \dots & m_{|\mathcal{T}||\mathcal{A}_s|} \end{bmatrix}$$

Using $\mathcal{G}$ ($\mathcal{G}_e$ or $\mathcal{G}_u$), a value of each element m_{ij} in the matrix is calculated according to Algorithm 2. When an app of the source user $a_{s,j}$ is also installed on the target user's smartphone, the mapping from $a_{s,j}$ to the target user app $a_{t,i}$ is calculated deterministically (lines 4–8) (For example, when $a_{t,i}$ is equal to $a_{s,j}$, $a_{s,j}$ is definitely converted into $a_{t,i}$ (line 6)). Otherwise, the mapping is calculated based on the similarity between $a_{t,i}$ and $a_{s,j}$ (lines 10–15). After $a_{t,i}$ and $a_{s,j}$ are converted into semantic word vectors (lines 11–12), $NormalizedSimilarity(v_{a_{t,i}}, v_{a_{s,j}})$ in the 13th line calculates the semantic similarity between $a_{t,i}$ and $a_{s,j}$ based on cosine similarity as follows:

$$NormalizedSimilarity(v_{a_{t,i}}, v_{a_{s,j}}) = \frac{Similarity(v_{a_{t,i}}, v_{a_{s,j}})}{S},$$

where

$$Similarity(v_{a_{t,i}}, v_{a_{s,j}}) = \frac{v_{a_{t,i}} \cdot v_{a_{s,j}}}{\left\|v_{a_{t,i}}\right\| \left\|v_{a_{s,j}}\right\|}$$

and $S = \sum_n Similarity(v_{a_{t,n}}, v_{a_{s,j}})$. That is, when apps $a_{t,i}$ and $a_{s,j}$ are similar with each other, m_{ij} has a large value. We calculate $\boldsymbol{M}$ for existing and unseen apps as $\boldsymbol{M}_e$ and $\boldsymbol{M}_u$, respectively. For $\boldsymbol{M}_e$, m_{ij} is zero when $a_{t,i}$ is an unseen app (line 15). This indicates that $a_{s,j}$ (which is not installed on the target user's smartphone) is converted only into existing apps. In contrast, for $\boldsymbol{M}_u$, m_{ij} is zero when $a_{t,i}$ is an existing app (line 15). This indicates that $a_{s,j}$ is converted only into unseen apps. In this manner, we can construct $\boldsymbol{M}_e$, which is used to convert an app of the source user not installed on the target user's smartphone only into an existing app. In addition, we can construct $\boldsymbol{M}_u$, which is used to convert an app of the source user not installed on the target user's smartphone only into an unseen app.

Algorithm 2: GenerateConversionMatrix

```
Input: $\mathcal{T}$: a set of apps from a target user, $\mathcal{A}$: a set of apps from a
       source user, $\mathcal{G}$: a set of unseen apps or existing apps
1  $\boldsymbol{M} \leftarrow |\mathcal{A}| \times |\mathcal{T}|$ zero matrix
2  for $i = 1$ to $|\mathcal{T}|$ do
3      for $j = 1$ to $|\mathcal{A}|$ do
4          if $a_{s,j} \in \mathcal{T}$ then
5              if $a_{t,i} = a_{s,j}$ then
6                  $m_{ij} \leftarrow 1$
7              else
8                  $m_{ij} \leftarrow 0$
9          else
10             if $a_{t,i} \in \mathcal{G}$ then
11                 $v_{a_{t,i}} \leftarrow$ Word-Embedding($a_{t,i}$)
12                 $v_{a_{s,j}} \leftarrow$ Word-Embedding($a_{s,j}$)
13                 $m_{ij} \leftarrow NormalizedSimilarity(v_{a_{t,i}}, v_{a_{s,j}})$
14             else
15                 $m_{ij} \leftarrow 0$
Output: Conversion matrix $\boldsymbol{M}$
```

4.3.2 Generating a Conversion Table

We generate a conversion table (dictionary) T probabilistically using $\boldsymbol{M}_e$ and $\boldsymbol{M}_u$ for each iteration, enabling us to create various training sequences toward robust next-app estimation. T describes a mapping from each source user app to a target user app. For each source user app a_s, we randomly select a conversion matrix $\boldsymbol{M}$: $\boldsymbol{M}_e$ or $\boldsymbol{M}_u$ (lines 16–20 in Algorithm 1). In this study, we use a large θ value to increase the number of usages of unseen apps to be included in the training sequences. Using $\boldsymbol{M}$, we determine a mapping from a_s to a target user app (line 22 in Algorithm 1) in accordance with Algorithm 3.

First, an app a installed on a source users' smartphone is represented using the 1-of-K scheme (line 1). Further, we compute the probability with which each target user app is converted from a in accordance with Eq. (1) (line 3). Finally, a target user app similar to a is selected probabilistically based on roulette wheel selection (line 4). In roulette wheel selection, an item is selected randomly according to the probability that is associated with each item. In this study, the probability of the i^{th} app (similarity between a and the i^{th} app) is stored in the i^{th} element of $\hat{\boldsymbol{p}}$. In addition, a mapping from a_s to a_t obtained by ConvertApp() is further added to T (line 24 in Algorithm 1).

4.3.3 Converting a Sequence of App Usages

Using the conversion table T, we convert the sequence of source user app usages $\mathcal{S}$ into a training sequence tailored to the target user (line 26 in Algorithm 1) in accordance with Algorithm 4. We convert the s^{th} usage of the source user app a_s into a target user app a_t using the conversion table T (line 4).

Algorithm 3: ConvertApp

```
Input: $\boldsymbol{M}$: a conversion matrix, $a$: an app from a source user
1  $\boldsymbol{o}_a \leftarrow$ 1-of-K($a$) /* 1-of-K representation of $a$ */
2  /* Compute conversion probabilities according to
     Equation 1 */
3  $\hat{\boldsymbol{p}} \leftarrow \boldsymbol{M}\boldsymbol{o}_a$
4  $\hat{a}_t \leftarrow$ Roulette-Wheel-Selection($\hat{\boldsymbol{p}}$)
Output: an app $\hat{a}_t$ similar to $a$
```

Algorithm 4: ConvertSequence

```
Input: $\mathcal{S}$: a sequence of app usages from a source user, $T$: a con-
       version table
1  $\mathcal{S}_t \leftarrow$ array with length $|\mathcal{S}|$
2  for $s = 1$ to $|\mathcal{S}|$ do
3      /* Obtain mapping from $a_s$ */
4      $a_t \leftarrow T[a_s]$
5      $\mathcal{S}_t[s] \leftarrow a_t$
Output: a sequence of app usages tailored to a target user $\mathcal{S}_t$
```

4.4 Predicting a Next-use App Using a Neural Network

In accordance with the procedure described in Section 4.3, we obtain multiple training sequences with length $k+1$ of app usages in which the $k+1^{th}$ usage is an answer, i.e., a next-use app. Thus, we train a prediction model to output the $k+1^{th}$ used app when a sequence of usages from 1 to k is provided as input. Each app usage is represented in the 1-of-K scheme with the number of dimensions being the number of apps installed on the target user's smartphone.

4.4.1 App Usage Time Interval

Here, the time interval of the i^{th} used app and $i+1^{th}$ used app in the sequences is not uniform. The time interval has a strong relation to the usage patterns of apps because an app used just after another app can relate strongly to the previously used app in many cases. For example, when a user is on a journey, he or she might first use a camera app to take a photo of a scenic spot, and then edit the photo with a graphics editor app. Finally, he or she will share the edited photo with friends using a social app, e.g., Facebook. The three different apps will be used in a short time and be strongly related. In contrast, when the time interval between apps in a sequence is too large, the apps might be entirely unrelated. Suppose that a user used a music app before going to sleep and then an e-mail app on the next morning. In this case, the two neighbor apps, the music and e-mail apps, are unrelated.

Figures 4, **5**, and **6** show the proportions of five categories of next-use apps for different time intervals when the last-used app belongs to the social category in our dataset. We find that apps belonging to some categories such as social apps are used frequently when the time interval is small. However, when the time interval increases, the proportions of the social category decrease, indicating that users tend to stop using social apps and then start using other kinds of apps when the time interval is large. In contrast, when the time interval increases, the proportions of game category apps increase. Furthermore, regardless of the length of the time interval, the proportion of the communication category apps is always larger than 50%, indicating that users always have an interest in using communication apps.

Figures 7, **8**, and **9** show the proportions of the five categories of next-use apps for different time intervals when the last-used app belongs to the communication category in our dataset. Interestingly, the proportions of apps belonging to the news and magazines category drastically decrease as the time interval becomes large. This means that users frequently check news and magazines just after they use communication apps. In contrast, game apps are frequently used when the time interval is large. In addition, apps belonging to the communication category are frequently used regardless of the time interval.

4.4.2 Interval-LSTM Model (*i*-LSTM Model)

Because the app usage history is time-series data, an LSTM network is a good choice for capturing the temporal features of the usage history. A traditional LSTM unit uses input, forget, and output gates to control the balance of saved information from previous input and new information from the current input. Our investigation, as described above, revealed that the time interval of app usages is an important factor in deciding whether to for-

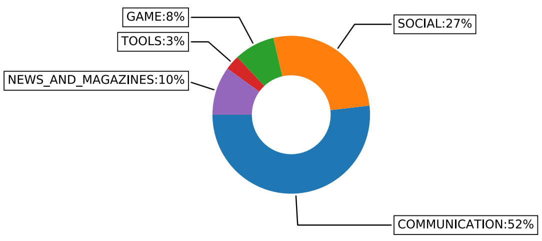

Fig. 4 Proportions of next-use app categories after a social app was used with a time interval of 0–10 min.

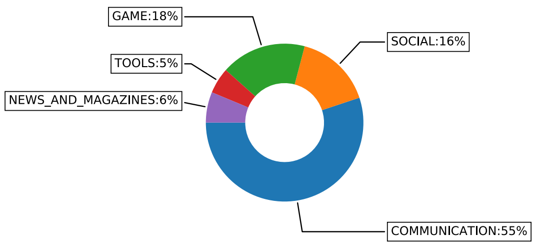

Fig. 5 Proportions of next-use app categories after a social app was used with a time interval of 10–30 min.

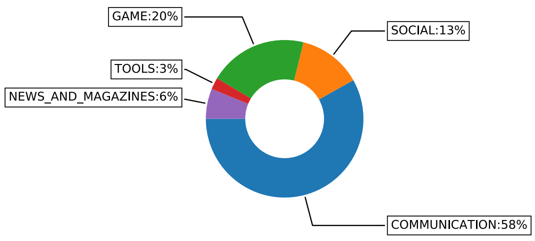

Fig. 6 Proportions of next-use app categories after a social app was used with a time interval of 30–60 min.

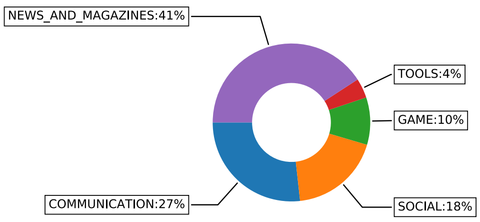

Fig. 7 Proportions of next-use app categories after a communication app was used with a time interval of 0–10 min.

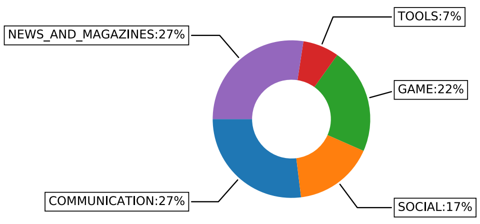

Fig. 8 Proportions of next-use app categories after a communication app was used with a time interval of 10–30 min.

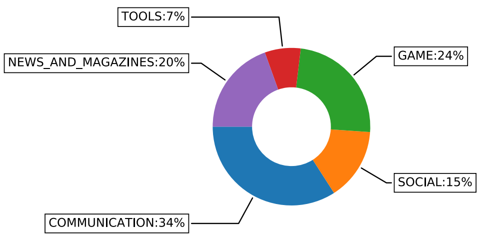

Fig. 9 Proportions of next-use app categories after a communication app was used with a time interval of 30–60 min.

get saved information or add new information in our task. When the time interval from the last used app is short, new information is more important in deciding what app to use next. In contrast, when the time interval from the last used app is long, saved information from previously used apps is less useful. However, the three gates of a traditional LSTM do not consider the time interval and regards all of the time intervals in the usage history as identical, preventing the modeling of app usages that relate strongly to the time interval. Consequently, we design a new LSTM variant considering the time interval by modifying the input and output gates as follows.

$$\begin{aligned} i_m &= \sigma_i(x_m W_{xi} + h_{m-1} W_{hi} + \Delta_m W_{ti} + b_i) \\ o_m &= \sigma_o(x_m W_{xo} + h_{m-1} W_{ho} + \Delta_m W_{to} + b_o) \end{aligned}$$

Here, i_m and o_m are the input gate's and output gate's activation vectors at time m, respectively. Furthermore, Δ_m is the time interval, x_m is the input usage vector of an app, σ_i and σ_o are sigmoid functions, h_{m-1} is the output vector at time $m-1$, W_{xi}, W_{hi}, W_{ti}, W_{xo}, W_{ho} and W_{to} are weight matrices, and b_i and b_o are biases. The traditional LSTM unit has a memory cell that holds past information and updates the cell state using write, read, and forget operations using the input, output, and forget gates, respectively. Therefore, the input gate decides which part of the information in the cell state is to be updated. In contrast, the output gate decides which part of the cell state we are going to output. Differently from a traditional LSTM unit, we add the influence of the time interval to the input and output gates. In our equations, Δ_m helps the input and output gates determine whether the model is to use more information from previous used apps or new input apps. The cell state c_m and output vector h_m are calculated as follows.

$$\begin{aligned} c_m &= (1 - i_m) \odot c_{m-1} + i_m \odot \tilde{c}_m \qquad (2) \\ \tilde{c}_m &= \sigma_c(x_m W_{xc} + h_{m-1} W_{hc} + b_c) \\ h_m &= o_m \odot \sigma_h(c_m) \end{aligned}$$

Here, W_{xc} and W_{hc} are weight matrices, b_c is a bias, and σ_c and σ_h are the tanh function. Because introducing the time interval increases the complexity of the model and is difficult to train, we use $1 - i_m$ in Eq. (2) instead of the forget gate in a traditional LSTM unit to reduce the complexity of the model inspired by Ref. [21].

4.4.3 Next-use App Prediction with *i*-LSTM

In accordance with the data structure of the training data, we adopt a two-layer many-to-one multi-label classification model [22] whose input and output are a sequence and a fixed-size vector, respectively. In our case, the input of the network is a series of app usage histories represented in the 1-of-K scheme, and the output is a vector consisting of the class probabilities of the respective apps, diversifying the predicted apps in a ranked list. The network consists of an *i*-LSTM layer with 400 nodes, a fully connected layer with 800 nodes with the ReLU activation functions, and an output softmax layer. To reduce overfitting, we use dropout, a simple regularization technique in which randomly selected nodes are ignored during training [23]. We train the network using backpropagation based on Adam [24] to minimize the categorical cross-entropy between estimates and the ground truth. In the prediction phase, an app usage sequence with length k obtained from the target user is provided as input to the network, and the network outputs the probability with which each app of the target user will be a next-use app. Finally, the apps with the top-N probabilities are chosen as the next-use app candidates.

4.5 Fine-tuning Neural Network

To alleviate the cold-start problems in next-use app prediction, we leverage the source users' usage histories to create training data for the neural network. However, the trained network can be regarded as a user-independent model and does not always capture the usage pattern of the target user. Because the app usage history of the target user has been accumulated after the user introduced the next-use app prediction system, we fine-tune the trained neural network using a low learning rate by leveraging the target user's accumulated usage history. The learning rate is a parameter that controls the rate at which the parameters are updated in a neural network. In our method, after the usage history of the target user for a particular period is accumulated, we update the model using the accumulated history.

5. Evaluation

5.1 Dataset

We collected an app usage dataset using our developed Android app called "context monster" [25]. Our developed app observes app usages, i.e., app launchings, and records application IDs as well as timestamps of app usage to be transmitted to a server computer.

Since some participants app usage history data are not enough or sparse, we selected 100 participants with the largest app usage history data as the experiment participants. Considering the participants privacy, before the experiment, we remove the user personal information (e.g., gender, age). Each participant in the dataset has 3,779 usage history data items on average. The average duration of the participants' data collection period was 68.6 days. The average number of installed apps was 60.7. The part of app category distribution in the dataset is shown in **Table 1**, the most used top 5 app categories are COMMUNCATION, GAME, SOCIAL, NEWS AND MAGAZINES and TOOLS. To obtain the app semantic representations, we retrieved a description of each app from Google Play. For apps that were not available on Google Play, we used the titles of the apps instead of descriptions.

Because the durations of data collected by some participants were not sufficiently long for performing fine-tuning, we selected top-twenty participants in terms of the length of the usage history, and we used their data to evaluate the proposed method. The remaining eighty participants' data are used only as training data.

Table 1 App category distribution.

Category	App usage percentage
COMMUNICATION	38.4%
GAME	16.8%
SOCIAL	14.8%
NEWS AND MAGAZINES	11.4%
TOOLS	8.5%
PRODUCTIVITY	3.0%
VIDEO PLAYERS	1.7%
PHOTOGRAPHY	0.9%

Table 2 Statistics of app usages by test participants.

	total # of apps	total # of usage sequences
Unseen apps	196	5,399
Existing apps	1,262	184,023
All apps	1,447	189,422

Each test participant had 9,430 usage history data items on average. The average duration of the test participants' data collection period was 82.5 days. The average number of installed apps was 52.8. **Table 2** presents the total number of unseen and existing apps and their usage sequences by the test participants. The usage sequence of an unseen app is the usage sequence with length k whose next-use app, i.e., the answer, is an unseen app ($k = 5$). In contrast, the usage sequence of an existing app is the usage sequence whose next-use app, i.e., the answer, is an existing app.

5.2 Evaluation Methodology

5.2.1 Evaluation Measure

To evaluate the app prediction methods, we used a top-N prediction accuracy metric widely used in studies on next-use app prediction [7], [10], [11], [12], [16]. If the *any* app in a set of N candidate apps is actually the next-use app, we regard the next-use app to be predicted accurately. The prediction accuracy metric is defined as **Accuracy@N** $= \frac{\sum_{i=1}^{T} H_i^N}{\sum_{i=1}^{T} A_i}$, where H_i^N is the number of accurately predicted next-use apps for a test user i, when N candidates are provided, A_i is the total number of test data for test user i, and T is the number of test users.

5.2.2 Methods

We evaluate the following methods to investigate the effectiveness of the proposed method.

- **MFU** (most frequently used): The top-N candidates are the most frequently used N apps in the training data.
- **RankSVM**: A ranking method proposed in Ref. [26], a learning-to-rank algorithm for query-based document search, is used to obtain the top-N next-use apps. RankSVM is a pair-wise ranking algorithm that computes a ranking list based on a pair of candidate items. In other words, the input to a support vector machine (SVM) is a pair of items, and the output is the probability that one item is ranked higher than another. In our case, each item corresponds to a target user app. Because RankSVM cannot deal with time series, we concatenate a vector of the latent next-use app and those of the previously used k apps and use the concatenated vector as an item. Note that each app is represented by an app semantic vector using our method.
- **One-hot**: The neural network architecture is identical to that of the proposed method. However, the training data are not tailored to a target user. Each app is represented in the 1-of-K scheme in which K is the size of the set of all apps installed on the smartphones of source or target users.
- **Word2vec**: This method uses a regression technique to output a semantic vector showing an estimate of a next-use app, as was briefly mentioned in the introduction. The neural network architecture is similar to that of the proposed method. Each app used in the input and output of the network is represented by a semantic app vector. In the prediction phase, we calculate the cosine similarity between an output vector and a semantic vector for each of the apps installed on a target user's smartphone. The top-N candidates are the N apps of the target user with the top-N cosine similarities. The loss function that is used to train the network is the mean squared error.
- **Proposed**: This is the proposed method.
- **Proposed (LSTM)**: This method uses traditional LSTM networks instead of i-LSTM networks.

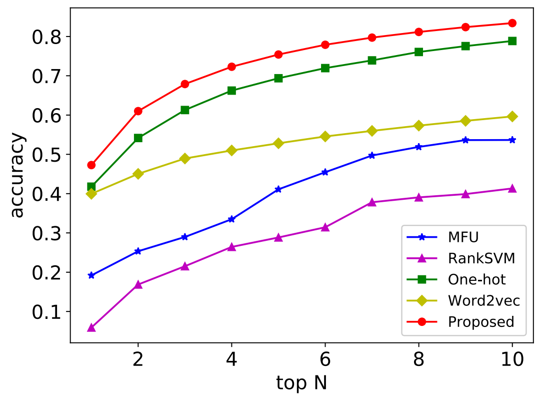

Fig. 10 Transitions of Accuracy@N when N is varied.

We evaluate the aforementioned methods based on 'leave-one-participant-out' cross validation. We consider one test participant to be the target user and the remaining nineteen test participants and eighty training participants as the source users.

5.2.3 Parameter Setting

For an app usage sequence that is provided as input to the neural network, we set the length of the sequence k to five.

For parameter θ in Algorithm 1, which is used to control the number of unseen apps to be included in the training data, we set θ to 0.7 according to the real ratio of unseen apps in the whole dataset. For the number of iterations P in Algorithm 1, which is used to control the number of training sequences tailored to a target user generated from a sequence obtained from a source user, we set P to ten. The dimension of each app semantic vector generated by the pre-trained Japanese Wikipedia word2vec model is set to 200. The number of epochs and batch size for neural network training are 10 and 1,024, respectively. These parameters are determined based on our preliminary experiment. An approximate time of 60 minutes was required to train a neural network for each participant using a server computer with NVIDIA Quadro P6000.

5.3 Results

5.3.1 Accuracy@N

Figure 10 shows the transitions of Accuracy@N for the various methods when N is varied. Note that fine-tuning was not performed for Proposed. We will investigate the effects of fine-tuning later. As shown in the figure, Proposed achieves the best performance. The poor performance of MFU indicates the difficulties associated with next-use app prediction. Because the apps used by the participants are diverse, it is difficult to predict the next-use apps using only information related to the frequency of usage. The performance of RankSVM is poor because the model cannot deal with time-series data. In addition, the high dimensionality of the model's input vectors can also degrade performance. The Accuracy@N for Word2vec does not exhibit a substantial alteration when N is varied. This result indicates that the candidate apps provided by Word2vec are not diverse. Word2vec outputs a semantic vector and selects candidate apps according to the similarity between an app's semantic vector and the output vector, resulting in the low diversity of the app candidates, i.e., including only apps similar to the output vector. Therefore, although Word2vec achieves good performance when N is one, the performance does not increase as N increases. As above, even when we used semantic representations of apps, we could confirm that the simple regression-based approach does not work well in this task. Even when N is one, Proposed outperforms Word2vec because Proposed generates a variety of training data tailored to target users from app usage data from source users. Proposed achieves the optimal performance and outperforms One-hot by approximately 5%.

Figure 11 shows the Accuracy@N of the methods for the existing apps. The figure shows that while Proposed outperforms One-hot by approximately 4% when N is small, the performances of Proposed and One-hot are nearly identical when N is large. This is because the neural network architectures of these methods are identical. However, One-hot cannot capture the relationship between usages of unseen and existing apps, slightly degrading its performance.

As shown in **Fig. 12**, with respect to the Accuracy@N of unseen apps, Proposed considerably outperformed the other methods. Surprisingly, Proposed achieved 54% accuracy when $N = 10$ even though the usage history of these unseen apps by source users is not available at all. One-hot and MFU could not predict the use of unseen apps at all as a result of their architectures. While their accuracies are poor, RankSVM and Word2vec could also predict the usage of these unseen apps, because these methods compute app candidates based on the semantics of apps. However, Proposed, the architecture based on the 1-of-K scheme, works effectively for this multi-class classification task. Interestingly, even when N is one, Proposed significantly outperforms Word2vec. This might be because Proposed generates a variety of training data tailored to target users. Because usages of unseen apps are unpredictable, the diverse training data generated by Proposed greatly contributes to the method performance.

5.3.2 Effect of *i*-LSTM

We investigate the effect of our *i*-LSTM model on the next-use app prediction task. Here, we divide next-use apps that we want to predict into two types, as follows:

- **Short-term next-use app**: The time interval between a last-used and next-use app that we want to predict is shorter than K minutes.
- **Long-term next-use app**: The time interval between a last-used and next-use app is equal to or longer than K minutes.

As described in Section 4.4.1, usage patterns of apps are strongly dependent on their categories. **Figures 13** and **14** show the transitions of Accuracy@N when N varies ($K = 10$). In general, the long-term next-use apps are more difficult to predict than the short-term next-use apps because they have a weak relation to previously used apps and more randomness. As shown in

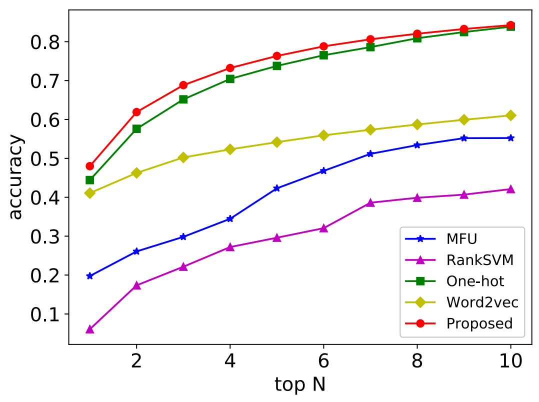

Fig. 11 Transitions of Accuracy@N for existing apps when N is varied.

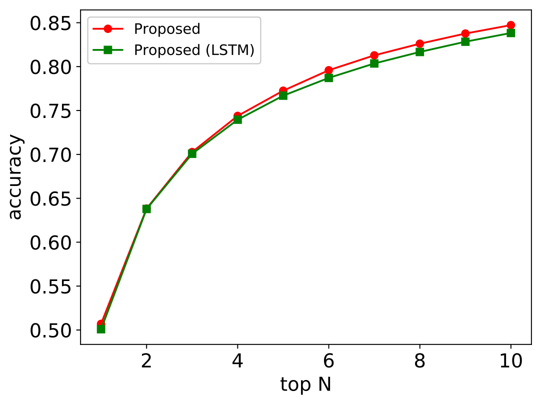

Fig. 13 Transitions of Accuracy@N for short-term next-use apps.

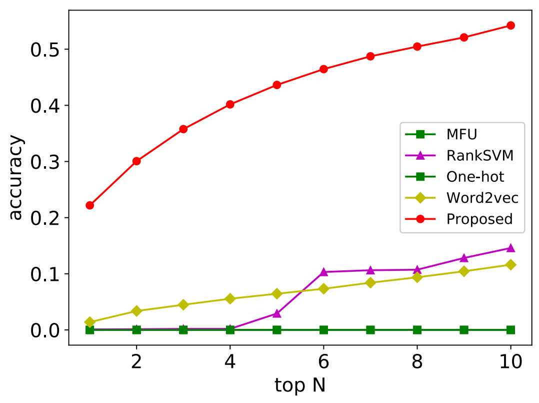

Fig. 12 Transitions of Accuracy@N for unseen apps when N is varied.

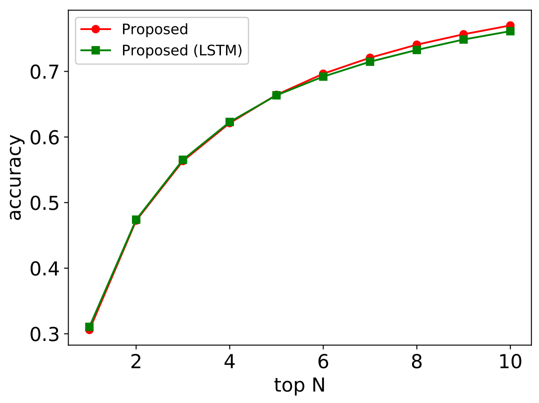

Fig. 14 Transitions of Accuracy@N for long-term next-use apps.

the figures, although the performances of Proposed and Proposed (LSTM) do not differ when N is small, Proposed outperforms Proposed (LSTM) by approximately 1% for both the short- and long-term next-use apps, indicating the advantage of the i-LSTM model that takes into account the interval of app usages.

5.3.3 Impact of Fine-tuning

Here, we investigate the impact of fine-tuning when fine-tuning was performed every half month. **Figure 15** shows the Accuracy@N when we perform fine-tuning, showing approximately 1% improvement as a result of fine-tuning. Specifically, Accuracy@N for unseen apps improves by approximately 5%, as shown in **Figs. 16** and **17**, meaning that 62% unseen apps was predicted successfully when $N = 10$. Because actual usage histories of unseen apps are unavailable in the usage histories of source users, fine-tuning greatly improves Accuracy@N for unseen apps.

Figure 18 shows the transitions of the accuracies when fine-tuning was performed and not performed. The vertical axis of the figure depicts the mean accuracy over Accuracy@N (N ranges from one to ten). As depicted in Fig. 18, the effect of fine-tuning was limited when the total quantity of additional training data that were used for fine-tuning was small ($\leq$ 60 days). In particular, when only half-month or one-month fine-tuning data were used, fine-tuning deteriorates the performance. This might be because the overfitting caused by these fine-tuning data. In contrast, when the total quantity of additional training data used for fine-tuning was sufficient (> 60), the improvement caused by fine-tuning was approximately 2%. The use of unseen apps is estimated on the basis of only the usage patterns of semantically similar apps. Addition of the actual usage history of the unseen apps by a target user as training data improved the prediction accuracies of the unseen apps.

5.3.4 Effects of P

Figure 19 depicts the Accuracy@N for Proposed when P in Algorithm 1 was varied. In addition, **Figs. 20** and **21** depict the Accuracy@N of existing and unseen apps, respectively, for Proposed when P was varied. Basically, the accuracies increased as larger P values were used. In particular, the accuracies of the existing apps when $P = 5$ were much higher than those obtained when $P = 1$. In contrast, because the accuracies for the existing apps when $P = 10$ are not very different from those when $P = 5$, the P value that yields the upper bound of the accuracy can be considered to be $P = 10$. The accuracies of the unseen

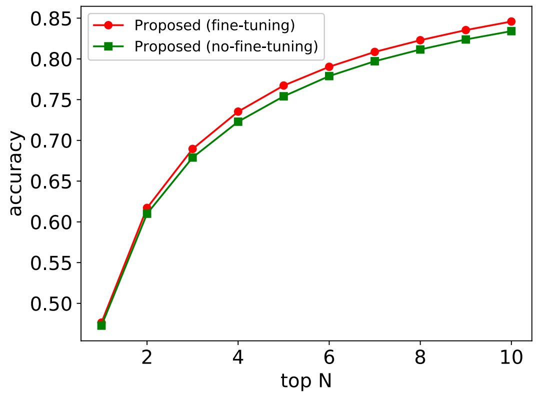

Fig. 15 Transitions of Accuracy@N when fine-tuning is performed and when fine-tuning is not performed.

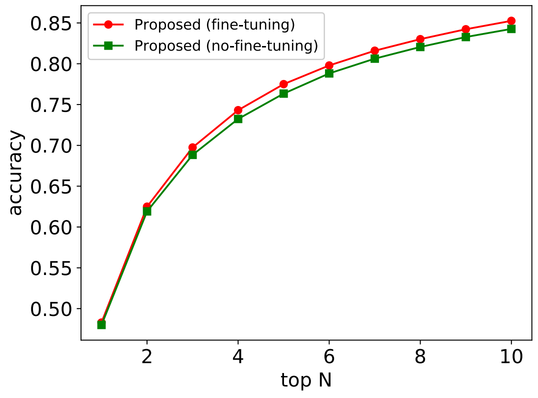

Fig. 16 Transitions of Accuracy@N for existing apps when fine-tuning is performed and when fine-tuning is not performed.

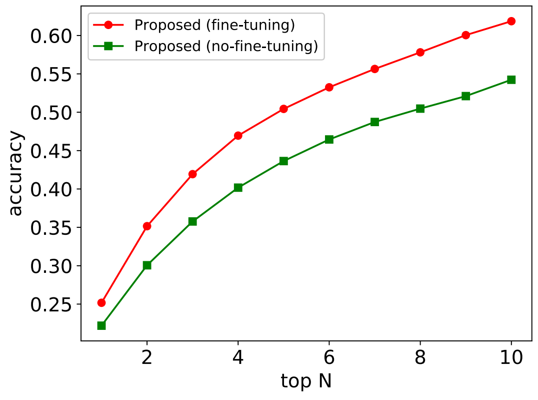

Fig. 17 Transitions of Accuracy@N for unseen apps when fine-tuning is performed and when fine-tuning is not performed.

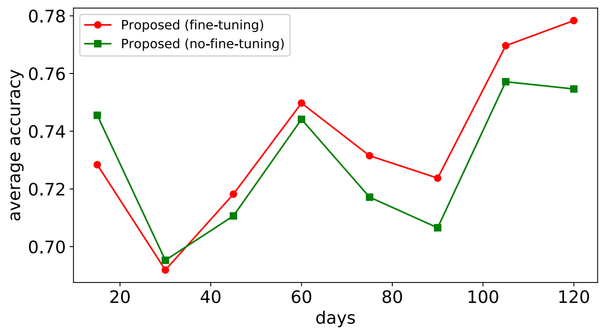

Fig. 18 Transitions of average accuracies for Proposed when fine-tuning is performed and when fine-tuning is not performed.

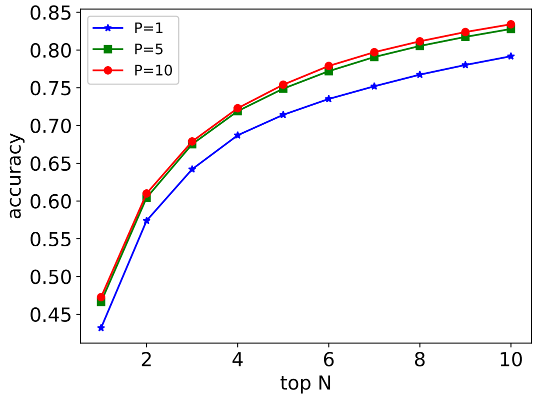

Fig. 19 Transitions of Accuracy@N for Proposed when P is varied.

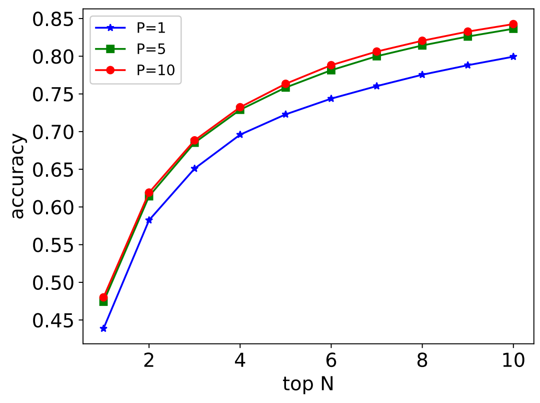

Fig. 20 Transitions of Accuracy@N of existing apps for Proposed when P is varied.

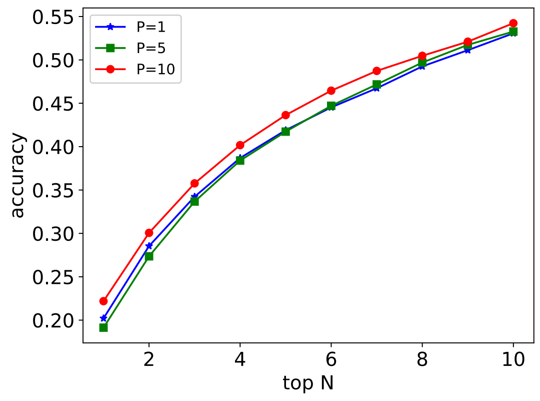

Fig. 21 Transitions of Accuracy@N of unseen apps for Proposed when P is varied.

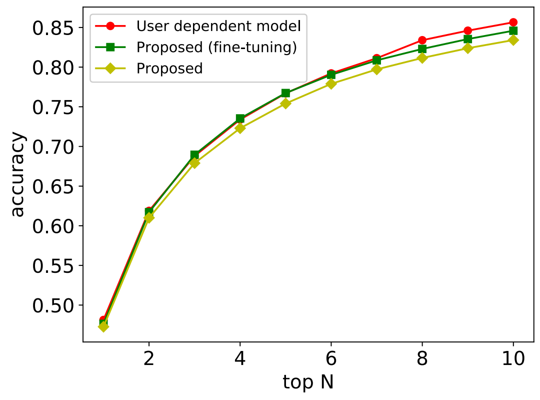

Fig. 22 Transitions of Accuracy@N for the user-dependent and Proposed models.

apps when $P = 10$ are somewhat higher than those that are observed when $P = 5$. This is because the quantity of training data for the unseen apps was not sufficient when $P = 5$.

5.3.5 User-dependent Model

We investigated the prediction performance of a user-dependent model that was trained on the first half of the usage history of a participant and then tested it using the latter half for the same participant. **Figure 22** shows the transitions of Accuracy@N with varying N. The performance of the user-dependent model can be regarded as the upper-bound of next-app prediction. Surprisingly, the performance difference between the user-dependent model and Proposed (fine-tuning) is almost identical when $N < 8$. In addition, the performance difference between the user-dependent model and Proposed is only approximately 1%. This means that the performance of the proposed method is nearly the same as that of the case in which training data for target users are fully provided.

5.4 Application and Limitation

The proposed method is expected to apply in smartphone to help the smartphone makers improving the service quality. The proposed method can benefit both smartphone developers and users. For smartphone developers, the proposed method helps them to pre-load the potential next-use app and close the inactive background apps to improve the efficiency of system. The smartphone developers can choose the top-N next-use app candidates by our proposed method as the background apps. The real next-use app has over 80% probability be chosen as the background app when 10 candidates are given by the proposed method. Furthermore, even the new installed apps without usage history have 62% probability can be selected as the background app when 10 candidates are given, comparing to 20% accuracy of random guess (random select 10 apps from 50 average installed apps), proposed method helps to allocate resource of smartphone efficiently. For smartphone users, the proposed method assists them selecting apps effectively. The large number of installed apps leads the app selection taking a lot of time. To help user select the next-use app, many smartphones hint the next-use app candidates when a user tries to open a app. Our proposed method makes the real next-use app has over 80% probability included in next-use app candidates and even the new installed apps without usage history have 62% probability successfully selected when 10 candidates are given, which improves user experience. Compare to the existing methods, our method achieves the same level accuracy in the next-use app prediction problem. Additionally, the most of existing methods such as Ref. [11] are hard to predict the new installed apps without history data, our method still achieves high accuracy to predict the new installed apps. However, when a new app is installed in a smartphone, the model of the proposed method has to be trained again to let the new installed app can be predicted. The retraining process may acquire long time and large computing resource. In the future, we would like to improve the proposed method to let the proposed method does not need to retrain the model when a new app installed.

6. Conclusion

This study proposed a new method for predicting next-use mobile apps based on the app usage history of a target user using the training data collected from other users (source users). The proposed method makes use of the semantic representations of apps to predict a target user's usage of apps that are not installed on the smartphones of source users. Our experiment, conducted using the actual app usage data, revealed that the proposed method achieved an accuracy of 62% (Accuracy@N; $N = 10$) when predicting the usage of apps that were not installed on the smartphones of source users.

Acknowledgments This work has been partially supported by JST CREST Grant Number JP-MJCR21F2.

References

[1] Baeza-Yates, R., Jiang, D., Silvestri, F. and Harrison, B.: Predicting The Next App That You Are Going To Use, *8th ACM International Conference on Web Search and Data Mining* (*WSDM '15*), pp.285–294 (2015).

[2] Cao, H. and Lin, M.: Mining smartphone data for app usage prediction and recommendations: A survey, *Pervasive and Mobile Computing*, Vol.37, pp.1–22 (2017).

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

[4] Yan, T., Chu, D., Ganesan, D., Kansal, A. and Liu, J.: Fast App Launching for Mobile Devices Using Predictive User Context, *10th International Conference on Mobile Systems, Applications, and Services* (*MobiSys '12*), pp.113–126 (2012).

[5] Natarajan, N., Shin, D. and Dhillon, I.S.: Which App Will You Use Next?: Collaborative Filtering with Interactional Context, *7th ACM Conference on Recommender Systems* (*RecSys '13*), pp.201–208 (2013).

[6] Sun, C., Zheng, J., Yao, H., Wang, Y. and Hsu, D.F.: AppRush: Using Dynamic Shortcuts to Facilitate Application Launching on Mobile Devices, *ANT/SEIT* (2013).

[7] Zou, X., Zhang, W., Li, S. and Pan, G.: Prophet: What App You Wish to Use Next, *2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication* (*UbiComp '13*), pp.167–170 (2013).

[8] Bishop, C.M.: *Pattern Recognition and Machine Learning* (*Information Science and Statistics*), Springer (2006).

[9] Hochreiter, S. and Schmidhuber, J.: Long Short-Term Memory, *Neural Comput.*, Vol.9, No.8, pp.1735–1780 (1997).

[10] Liao, Z.-X., Pan, Y.-C., Peng, W.-C. and Lei, P.-R.: On Mining Mobile Apps Usage Behavior for Predicting Apps Usage in Smartphones, *22nd ACM International Conference on Information & Knowledge Management* (*CIKM '13*), pp.609–618 (2013).

[11] Shin, C., Hong, J.-H. and Dey, A.K.: Understanding and Prediction of Mobile Application Usage for Smart Phones, *2012 ACM Conference on Ubiquitous Computing* (*UbiComp '12*), pp.173–182 (2012).

[12] Huang, K., Zhang, C., Ma, X. and Chen, G.: Predicting Mobile Application Usage Using Contextual Information, *2012 ACM Conference on Ubiquitous Computing* (*UbiComp '12*), pp.1059–1065 (2012).

[13] Laurila, J.K., Gatica-Perez, D., Aad, I., Bornet, O., Do, T.-M.-T., Dousse, O., Eberle, J., Miettinen, M., et al.: The Mobile Data Challenge: Big Data for Mobile Computing Research, *Mobile Data Challenge Workshop* (*MDC*) *in Conjunction with Pervasive* (2012).

[14] Liao, Z.-X., Li, S.-C., Peng, W.-C., Philip, S.Y. and Liu, T.-C.: On the Feature Discovery for App Usage Prediction in Smartphones, *2013 IEEE 13th International Conference on Data Mining* (*ICDM '13*), pp.1127–1132 (2013).

[15] Wang, Y., Yuan, N.J., Sun, Y., Zhang, F., Xie, X., Liu, Q. and Chen, E.: A Contextual Collaborative Approach for App Usage Forecasting, *2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (*UbiComp '16*), pp.1247–1258 (2016).

[16] Xu, Y., Lin, M., Lu, H., Cardone, G., Lane, N., Chen, Z., Campbell, A. and Choudhury, T.: Preference, Context and Communities: A Multi-faceted Approach to Predicting Smartphone App Usage Patterns, *2013 International Symposium on Wearable Computers* (*ISWC '13*), pp.69–76 (2013).

[17] Lin, J., Sugiyama, K., Kan, M.-Y. and Chua, T.-S.: Addressing Cold-start in App Recommendation: Latent User Models Constructed from Twitter Followers, *36th International ACM SIGIR Conference on Research and Development in Information Retrieval* (*SIGIR '13*), pp.283–292 (2013).

[18] Kudo, T., Yamamoto, K. and Matsumoto, Y.: Applying conditional random fields to Japanese morphological analysis, *2004 Conference on Empirical Methods in Natural Language Processing* (*EMNLP '04*), pp.230–237 (2004).

[19] Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S. and Dean, J.: Distributed Representations of Words and Phrases and their Compositionality, *Advances in Neural Information Processing Systems 26*, pp.3111–3119 (2013).

[20] Maaten, L.v.d. and Hinton, G.: Visualizing data using t-SNE, *Journal of Machine Learning Research*, Vol.9, No.Nov, pp.2579–2605 (2008).

[21] Zhu, Y., Li, H., Liao, Y., Wang, B., Guan, Z., Liu, H. and Cai, D.: What to Do Next: Modeling User Behaviors by Time-LSTM, *26th International Joint Conference on Artificial Intelligence*, *IJCAI-17*, pp.3602–3608 (2017).

[22] Lipton, Z.C., Kale, D.C., Elkan, C. and Wetzel, R.C.: Learning to Diagnose with LSTM Recurrent Neural Networks, *CoRR*, Vol.abs/1511.03677 (online), available from ⟨http://arxiv.org/abs/1511.03677⟩ (2015).

[23] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, *Journal of Machine Learning Research*, Vol.15, pp.1929–1958 (2014).

[24] Kingma, D.P. and Ba, J.: Adam: A Method for Stochastic Optimization, *International Conference on Learning Representations* (*ICLR*) (2015).

[25] Mizuno, S., Osawa, J., Hara, T. and Nishio, S.: A Preference Extraction Method Using Installed Applications and Their Descriptions, *2014 IEEE 33rd International Symposium on Reliable Distributed Systems Workshops*, pp.64–69 (2014).

[26] Joachims, T.: Optimizing Search Engines Using Clickthrough Data, *8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (*KDD '02*), pp.133–142 (2002).

Cheng Chen received his master degree from Graduate School of Information Science and Technology, Osaka University. His research interests include ubiquitous and time series forecasting.

Takuya Maekawa is an associate professor at Osaka University, Japan. His research interests include ubiquitous and mobile sensing. He received his doctor degree (Information Science and Technology) from Graduate School of Information Science and Technology, Osaka University. He serves as a director of IPSJ (2021–).

Daichi Amagata is an assistant professor with the Department of Multimedia Engineering Graduate School of Information Science and Technology Osaka University, Osaka, Japan. He received his B.E., his M.Sc., and his Ph.D. degrees from Osaka University in 2012, 2014, and 2015, respectively. His research interests include fast algorithms for query processing and data mining. He is a member of IEEE.

Takahiro Hara received his B.E., M.E., and Dr.E. degrees in Information Systems Engineering from Osaka University, Osaka, Japan, in 1995, 1997, and 2000, respectively. Currently, he is a full Professor of the Department of Multimedia Engineering, Osaka University. His research interests include databases, mobile computing, social computing, WWW, and wireless networking. He served and is serving as an associate editor of a number of international journals such as IEEE Transactions on Vehicular Technology, IEEE Open Journal of the Computer Society, and IEEE Access. He is a distinguished scientist of ACM, a senior member of IEEE, and a member of IEICE.