XRec: Behavior-Based User Recognition Across Mobile Devices

XIAO WANG, TONG YU, MING ZENG, and PATRICK TAGUE, Carnegie Mellon University

As smartphones and tablets become increasingly prevalent, more customers have multiple devices. The multi-user, multi-device interactions inspire many problems worthy of investigation, among which recognizing users across devices has significant implications on recommendation, advertising and user experience. Unlike the binary classification problem in user identification on a single device, cross-device user recognition is essentially a set partition problem. The app back-end aims to divide user activities on devices hosting the app into groups each associated with one user. In this paper, we present XRec which leverages user behavioral patterns, namely when, where and how a user uses the app, to achieve the recognition. To address the user-device partition problem, we propose a classification-plus-refinement algorithm. To validate our approach, we conduct a field study with an Android app. We instrument the app to collect usage data from real users. We provide proof-of-concept experimental results to demonstrate how XRec can provide added value to mobile apps, with the ability to correctly match a user across multiple devices with 70% recall and 90% precision.

CCS Concepts: •**Human-centered computing** →**Empirical studies in ubiquitous and mobile computing;** •**Information systems** →*Information systems applications; Data mining;* •**Computing methodologies** →*Machine learning;*

ACM Reference format:
Xiao Wang, Tong Yu, Ming Zeng, and Patrick Tague. 2017. XRec: Behavior-Based User Recognition Across Mobile Devices. *PACM Interact. Mob. Wearable Ubiquitous Technol.* 0, 0, Article 1 (May 2017), 26 pages.
DOI: 0000001.0000001

1 INTRODUCTION

The last decade has witnessed the emergence and rapid spread of mobile electronic devices such as smartphones and tablets. These new technologies have continually worked their way into various dimensions of our individual lives. As the diversity and quantity of networked consumer products continue to grow, more and more customers are becoming multi-device users [16]. According to market research, 31% of US adults own both smartphones and tablets [27], UK households now own on average three different types of Internet-enabled devices [31], and 54% of tablet users share their device with other users [19]. From such studies, we see that (1) users switch between or simultaneously use more than one device and (2) devices are shared among multiple users.

The complexity of multi-device, multi-user interaction presents significant challenges to service providers that provide value-added services by collecting and analyzing user behavioral data, namely because there are no effective techniques to properly label the data with the active user of the device. If the service provider had a mechanism by which *to differentiate between multiple users on the same device* and *to match the app user across different devices*, however, this *user recognition* capability would provide significant improvements to advertising services, recommendation systems, and general user experience.

In the multi-device, multi-user scenario, we define a *session* as a single visit to the app by one user on one device. In the presence of device sharing, the app might observe sessions of multiple users on a shared device. By user recognition, an app vendor or service provider aims to attribute sessions on different devices to the same user.

DOI: 0000001.0000001

In essence, the app back-end system reveals underlying linkage of sessions on devices and their users. The system can partition the set of sessions of the app into groups, where each group includes the set of sessions involving one user during a particular time period of interest. Depending on whether the partitioning is conducted on a single device, the problem is further categorized as: (1) multi-user differentiation on the same device, and (2) user recognition across multiple devices. For the first problem, classification of multiple users sharing the same device can be approached using identification mechanisms proposed for the single-device scenario [4, 7, 18, 44]. Hence, to avoid reinventing the wheel, we concentrate our efforts on the second problem of cross-device recognition, assuming users of a single device are already segmented and can be treated individually. An example of the resulting cross-device user recognition concept is illustrated in Figure 1.

As a motivating example of cross-device recognition, consider a scenario in which a user browses the Yelp app on each of their mobile devices. For the sake of convenience and privacy, the user prefers not to log in to Yelp, since this is not forced by the app. In this case, Yelp cannot recognize the browsing activity as belonging to the same user across the multiple devices, and it misses an opportunity to gain valuable insight into the user's preferences and tendencies. At the same time, the user misses out on the potential added value that Yelp could provide in terms of personalized recommendations. In this case the user's choice not to log in negatively affects both parties' utility. However, Yelp may still be interested in finding ways to provide notions of personalization that do not impede on users' personal privacy, providing a win-win solution.

Based on the complexity of the multi-device landscape and varied user privacy and usability preferences, our primary goal in this work is to enable approximate user-session partitions in a way that provides added value to app vendors and service providers while also considering privacy and usability concerns of device users. Toward this goal, we begin by outlining the current state of the art in recognizing users in web and mobile systems.

Traditional approaches for user recognition rely on either user authentication or explicit tracking. On a single device, websites and apps may employ cookies to track a user's online activity [1, 42]. However, locally stored cookies cannot span devices to enable recognition in the cross-device scenario [30]. Native system identifiers such as Android's Advertising-ID and iOS's IDFA are limited to usage by different apps on a single device instead of cross-device tracking [32]. In addition, a system-level solution cannot traverse different mobile ecosystems (Android, iOS and so forth). Moreover, such solutions, like accessing user accounts saved locally on each device, bring severe privacy concerns of revealing the explicit identity to a third party. Even worse for the app, relying on device accounts may lead the provider to erroneously label the owner of the device as the user of the application, failing to realize that devices are shared by multiple users. Similarly, privacy-conscious users may opt to use different accounts on different devices, for example to keep their personal and corporate data separate to comply with policies. Hence, these current approaches are insufficient, and in many cases, the app will fail to recognize users across multiple devices.

To address the current limitation to cross-device attribution of user activities, we propose a behavior-based user recognition framework named *XRec* for the multi-device scenario. Our approach leverages behavioral data, namely where, when and how a user interacts with the app. XRec relies on several behavioral characteristics of users, which can collectively serve as an effective yet potentially anonymous way to link user sessions across devices. The behavioral characteristics include location (GPS readings, IP addresses), app-specific statistics, and touchscreen gesture patterns. XRec approaches this problem in two rounds of partitioning. First, XRec creates a course-grained partition of devices into small groups using location information. The intuition lays in the fact that devices of one user are likely to consistently collocate with each other. After this coarse-grained grouping, a user's devices are likely to be mixed with devices of family, friends and colleagues. Subsequently, within each group, XRec performs pairing based on behavior patterns such as touchscreen gestures and browsing behaviors, then refines the set of pairings into a proper partition.

The behavior-based approach has two advantages. First, *it requires no user effort beyond normal app usage.* Users can use apps as usual without explicitly identifying themselves. Second, *it is privacy-preserving as behaviors*

are bound to neither real identities nor explicit identifiers. Also, the behavioral patterns are specific to certain context, and can be used for identification only in particular applications. Hence, even leakage of the information does not expose user identity.

Our cross-device recognition approach distinguishes itself from previous behavior-based approaches for identification and authentication. Previous approaches have been proposed for identification and authentication on individual devices in which the mobile operating system aims to thwart unauthorized device access. The differences between these problems are significant. First, an operating system agent typically performs user authentication on a single device, while the *app back-end system* performs cross-device recognition in our approach. The two entities have their unique advantages and constraints in terms of behavior-related measurement collection. For example, the app back-end enjoys the advantage of obtaining app-specific usage patterns. Second, the user authentication is formalized as a binary classification problem with the owner's behaviors as positive instances. In cross-device recognition, however, we are trying to group devices that belong to the same user. The user's real identity is transparent to the app back-end and also of no interest. Hence, the cross-device recognition becomes a set partitioning problem, detailed in Section 3, that requires some algorithmic innovation to solve. Third, the authentication problem requires high accuracy since it is critical for security and privacy, while our approach to cross-device recognition is intended to add value to non-sensitive services such as advertising and user experience, so high accuracy is desirable but not necessary.

In our design and implementation of XRec, there are components of measurement collection, feature extraction and algorithmic inference. To present our approach in a more clear and solid fashion, we implement XRec using an Android app named Hacker News Reader [36], which is discussed in more detail in Section 3.2. Since it is the app back-end that attempts to recognize users, the provider can leverage exclusive usage information it collects through the app.

Summary of our work: In summary, our overall goal in this work is to *develop, test, and evaluate a proof-of-concept system for anonymously recognizing the same user across app usage sessions on multiple mobile devices.* To the best of our knowledge, this is the first attempt for multi-device user recognition without relying on explicit user authentication. Toward this goal, we provide the following contributions.

- We design the XRec protocol for anonymous, behavior-based user recognition across mobile devices based on application usage patterns and sensor measurements.
- We propose a classification-plus-refinement approach to identify user partitions within an observed set of sessions.
- We deploy XRec with an experimental Android app to collect and evaluate real usage data in a proof-of-concept experiment.

Roadmap. The paper is organized as follows. We first contrast our study with existing work. Next in Section 3, we describe background information about problem formulation and the experimental app. We then present in Section 4 our framework for cross-devices user recognition with details regarding technical assumptions, data collection, feature extraction and algorithm design. Section 5 and 6 present our experiment setup and results. In Section 7, we conclude and discuss future directions.

2 RELATED WORK

Incorporation of sensors into commodity mobile devices enables passive sensing for a variety of purposes, among which enormous research efforts have been devoted into device-based user identification. A device leverages sensory data to model users and distinguishes its owner from other users or thieves.

Researchers propose to use biometrics for user identification. Biometric approaches have been studied for many years, but only consider *user authentication on a single device* rather than multiple devices. Biometrics are categorized as physiological such as fingerprint and iris, and behavioral such as gait, keystroke, touch and

swipe [24]. Behavioral biometrics have the advantage of requiring no dedicated hardware [33]. Feng et al. [17] introduced a touchscreen-based approach that authenticates users through finger gestures. Vu [38] proposed an identification approach by exploiting capacitive touchscreens. Meng et al. [25, 26] focused on touch dynamics and experimented with touch biometrics. Frank et al. [18] and Xu et al. [41] investigated continuous and passive authentication based on how a user touches a screen. Jain et al. [21] developed an authentication approach by analyzing a user's behavioral traits modeled by acceleration data, curvature of swipes and finger area. Later, Li et al. [23] designed a mechanism to re-authenticate current users based on finger movements. Draffin et al. [15] modeled users' micro-behaviors during their interactions with a soft keyboard, including key press position, drift from finger-down to finger-up and touch pressure. Sae-Bae et al. [35] showed that multi-touch gestures are applicable to user authentication. De et al. [11] collected touchscreen input data and used dynamic time warping for identification. Dey et al. [13] leveraged accelerometer data to extract frequency features to obtain unique fingerprints. Zhu et al. [44] proposed a continuous authentication system that uses various sensors to profile users. This paper investigates user recognition across multiple devices that differs from authentication on a single device. *The problem introduces another dimension (device), and inherently requires multi-class user partitioning on multiple devices.* In contrast, previous research focuses on binary classification scenarios. In this sense, traditional approaches based on off-the-shelf classification algorithms cannot address the partitioning problem. We hence propose to leverage topological information to perform the cross-device recognition. To the best of our knowledge, similar approaches for partitioning have not been studied before.

Cross-device patterns received academic attention in the online search domain. Studies of usage patterns (topic of interests, reading behavior, etc.) have examined user behavior in the search engine. Wang et al. [39] examined cross-device task continuation from PC to smartphone for particular sets of search tasks. They used topics and transition time to predict whether a search task is the continuation from PC to mobile. Montañez et al. [28] studied search behaviors and device transition features to predict cross-device search transitions. Karlson et al. [22] analyzed the usage log of desktops and mobile phones to understand patterns of how people transition between devices. Our work investigates app usage behaviors across mobile devices. We believe there are still many research questions regarding how users and information transition between multiple devices.

3 PROBLEM DEFINITION

We formally define our problem through our example scenario, and highlight its distinction in contrast with the problem of user identification on a single device. We also introduce the experimental app for this study to assist with understanding of the problem and our approach.

3.1 Problem Statement

In our problem, an app's back-end aims to recognize the same users across multiple devices. The example scenario is illustrated in Figure 1. Each user uses the app on two devices as indicated by the lines. We approach the problem from the perspective that users are transparent to the app back-end. The back-end can only observe those devices and user interactions with the app. By recognizing users across devices, the app does not attribute a device to any user in reality. Instead, it essentially links devices together based on its inference whether they belong to the same user. In this sense, the output of our recognition algorithm is a *partition* of the set of devices, where a partition of a set is a division of the set's elements into non-empty subsets, each corresponding to a unique user. In the example scenario, the correct partition is $\{\{1, 5\}, \{2, 4\}, \{3, 6\}\}$.

For all devices, we further categorize them as *synced* and *unsynced*. A device is regarded as synced if it is linked to at least one of the other devices through "out-of-band" information. The information is usually user logins as some users may log in to the app on those devices. Otherwise, in the extreme case that no users opt to log in to the app, app developers may recruit people to use the app on multiple devices to manually create synced devices. In practice, those synced devices provide training samples for our algorithms. In Figure 1, device

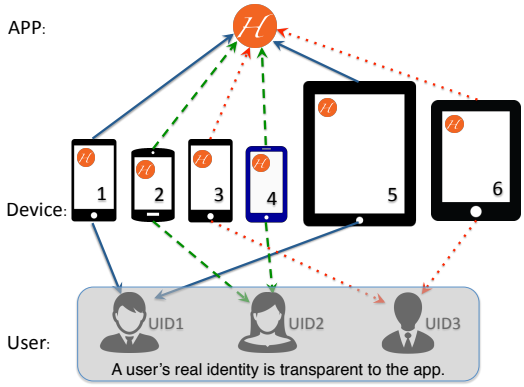


Fig. 1. We illustrate the scenario of user recognition across devices: associating devices that belong to a same user. Solid lines indicate synced devices whose association is already known to the app while dashed lines present unsynced devices to be associated using XRec. This example assumes no sharing of any device by multiple users. In the presence of sharing, the recognition is performed at the granularity of sessions instead of devices.

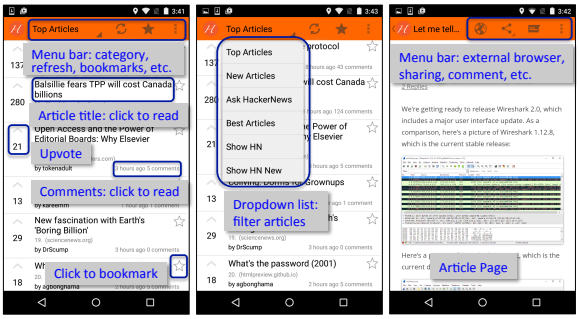


Fig. 2. Our experimental app is a news reader with diverse functionality including article filtering and bookmarking. It thus enables ample interaction with users, creating opportunities of obtaining effective behavioral patterns. The app also demonstrates key design components of mobile apps such as the list and menu layout, which helps generalize our ideas to many other apps.

1 and 5 are synced possibly because UID1 logs in on both devices while other devices are unsynced since users opt not to log in. If UID1 only logs in on device 1, then both 1 and 5 are unsynced. We have to notice that synced devices might be further linked to other devices. In reality, it is possible that a user logs in only on some devices or uses different accounts.

3.2 The Experimental App

Some previously published works study continuous authentication on single devices through touchscreen analytics. To collect measurements from users, they employ their own experimental app with particularly designed user interfaces. For example, in *touchalytics* [18], experiment subjects are instructed to use a simple image comparison app. They have to move the screen content through some touchscreen gestures in order to navigate between images. In another work by Xu et al. [41], the authors design an experimental app that instructs

participants to perform predefined operations such as typing in a sentence using the soft-keyboard and writing down a character on the screen.

In this paper, we instead *make use of a real-world app to avoid constraints in controlled settings*. Moreover, the touchscreen gestures consequently incorporate contextual information of the design and content of the app. That means we can analyze the gestures with respect to the app context instead of purely analyzing isolated gestures. For instance, a user's click locations may exhibit certain patterns due to the layout of the user interface.

To this end, we create a custom-instrumented version of the open-source Hacker News app [36] which is a popular mobile client for browsing YC Hacker news [29]. This app manifests typical design principles of mobile apps on the market. For instance, it employs the list-plus-menu-bar layout that is especially suitable for content display and page transition on small-screen devices. As shown in Figure 2, the app lists news articles for users to browse. Moreover, it enables functionality such as article filtering by category, upvoting/downvoting, sharing with friends and bookmarking. With all the diverse functionality, the app enables ample interaction with users, creating opportunities to identify useful patterns among users for the recognition purpose. Our version of the app measures context and behavior and reports back to our server.

This example can facilitate our explanation of the problem context and the intuition behind our proposed algorithms. Although our exposition is specific to this app, the framework and methodology are not limited to this app; instead the app serves as a proof-of-concept implementation using common design principles of mobile apps. Additionally, we will leverage app-independent features that are not associated with the design of an app. We admittedly notice that one single app cannot be representative of all apps and possible interactions with users. However, with the aforementioned reasons, we believe our approach to be generalizable to a broad class of apps. Again, XRec provides cross-device user recognition as added-value, and the inability to support the requirements simply means the app cannot enjoy the benefit of cross-device recognition brought by our approach and all device users would be treated independently.

4 DESIGN AND IMPLEMENTATION

In this section, we present our design and implementation of XRec with the experimental app. We give examples how user behaviors offer identifiable information and help with cross-device recognition. As shown in Figure 3, XRec contains indispensable components of measurement collection, data preparation and algorithmic inference. In what follows, we will detail each component.

4.1 App Instrumentation and Measurement Collection

We instrument the Hacker News app [36] and set up a server to collect usage-related measurements. Data are first stored locally on the device before being uploaded to the server. To reduce run-time overhead, the instrumented code only runs when the app is open, never in the background. Moreover, data are uploaded through WiFi networks to avoid cellular data usage.

The instrumented code collects time-stamped measurements about user actions. To model a user's usage pattern, the app records the user's interactions with the app including clicks, swipes and other actions that change the status of the app. Ideally, the sequence of measurements will serve as a behavioral fingerprint that is sufficiently different from that of other users. Since measurement collection relies on specific features we desire, particulars will be presented in the next subsection with discussions about potential features.

To obtain ground truth information for evaluating our approach experimentally, we require users to register or provide login credentials the first time the app runs on a new device. In this way, we are aware of which user uses the app on which devices and are able to obtain labeled data for our experiment.

In a real deployment, app developers have to consider *availability* of desired measurements, *cost* of implementation, run-time *overhead* and *privacy* concerns [3, 41, 43].

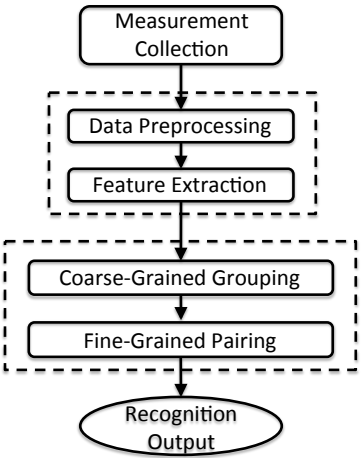


Fig. 3. This flowchart presents necessary components and steps of XRec for cross-device user recognition.

Server and device logs are easy to obtain. Some sensors may not exist on certain devices. Moreover, sensor accessibility may require user acknowledgement. It is preferred to use sensor permissions that are inherently granted to the app for its normal functionality. If the app requires additional sensor permissions for the purpose of user recognition, privacy-cautious users may decline it. For example, Yelp inherently needs GPS data to fulfill its services. The GPS info can be leveraged for user recognition. However, if the app further requests for irrelevant permissions such as contacts, it may raise a red flag. On the other hand, user acknowledgement is exempted for some permissions. For example, in Android, many permissions are designated as `protection_normal` such as access to coarse-grained location, network state and WiFi state. If an app requests a normal permission in its manifest, the system automatically grants it at install time while users are not prompted to manually grant the permission, and users cannot revoke it.

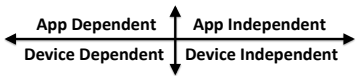
Effort of developers is another factor to consider. Collecting certain measurements can be demanding and less desirable. Preferably, developers are inclined to measurements that are inherently used by the app for its normal functionality and measurements that can be easily collected.

Developers need to bear in mind the run-time overhead associated with measurement collection. It is supposed to be non-invasive. We prefer it not to interfere with daily use of the app. Compared to systems that attempt to provide continuous authentication, user recognition is not a frequent service. Measurement collection can thus be disabled most of time. In this sense, any negative effects it may impose to app performance are significantly contained.

Developers may take user privacy into consideration. Collection of any explicit user identifiers is privacy-invasive. The amount and types of data to be measured should not exceed the minimum needed for linking multiple devices to a same user instead of identifying any users in the real world.

4.2 Feature Preparation

To obtain sufficient entropy for distinguishing different users, we consider a variety of features that cover various dimensions of a user's usage patterns. We hope the features can collectively model user behaviors and facilitate cross-device recognition. Generally, the features fall into different categories as shown below determined by whether the features are stable across apps or devices.



The multi-device scenario presents new challenges since some users might have different behaviors on different devices in terms of certain features. *This requires us to include features from different aspects of user behaviors, each of which contributes some entropy for identification. In the worst case, however, we cannot rule out the possibility that some users might act differently on different devices measured by all these features. Then for these users, it is difficult for the app to gain the added value offered by our approach, which is not favorable but not critical, either.* To reduce the possibility of this case, it is important to incorporate device-independent features that maintain permanence across devices, or exhibit relatively constant difference between different devices.

We also categorize features based on their app dependence. App-dependent features are created with respect to the context of the app. App-independent features do not rely on specific apps, and thus can be directly applied to other apps. Nevertheless, ideas in creating app-dependent features also applies to a broader class of apps.

In the following, we discuss several categories of features, and illustrate data from real users to facilitate understandings.

4.2.1 Session Characteristics. We leverage session-level discrepancy among different users. Each session represents a user's activities during a single visit to the app. The activities captured by our instrumented code are all time-stamped. We are able to divide the sequence of activities into sessions. For each session, we compute the following statistics.

(*stat-1*). Number of articles read per session
(*stat-2*). Time spent on each article
(*stat-3*). Number of sessions during each hour of a day

In Figure 4, we show the above session characteristics of three randomly chosen users on two types of devices: Nexus 4 phone and Nexus 7 tablet. Within sub-figures, (a-1) and (a-2) plot the CDF of ***stat-1***. As it shows, one user tends to read only a few articles per session while another consumes significantly more. They maintain their habits across two devices except for one user who exhibits less consistency. In sub-figures (b-1) and (b-2), ***stat-2*** manifests itself differently for three users: one user takes more time to read the articles than the other two. This trend also migrates across devices. Sub-figures (c-1) and (c-2) depict preference over 24 hours in a day for using the app on each device. Phones gain popularity at noon and evening while tablets are more preferred at night. All session statistics provide entropy to distinguish different users on multiple devices.

4.2.2 Clickstream Analysis. We zoom in to investigate clicks within each session. Each click typically triggers certain functionality of the app and enables navigation within the app. We can leverage differences in click ordering to model usage behaviors. The click sequence helps characterize how a user transitions from one activity to another. Formally, we use a Markov Chain model to analyze click transitions of a user. The Markov transition matrix can be directly used as a feature for classification.

In this model, each state is an app state, and edges represent transitions between app states triggered by user actions such as touchscreen clicks. Our instrumented code listens on `onClick` events, and records detailed information about each click including time-stamp and what activities are triggered.

For better comparison, we visualize transition matrices as heat maps in Figure 5. Inspecting all sub-figures, column-wise they show three users have different transition patterns. Row-wise, the same user maintains similar behavior on both devices. This feature is device-independent for many users.

4.2.3 Clicks and Swipes. Swipes and clicks, as illustrated in Figure 6, are important gestures to interact with apps. In our app, a user has to swipe on the screen to scroll the list of articles, and click an article title to read it.

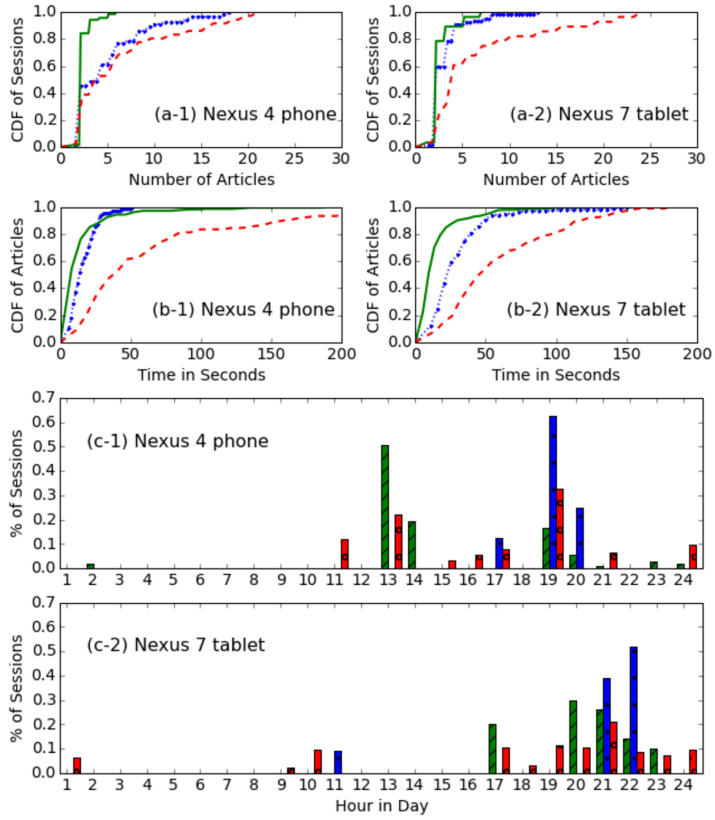


Fig. 4. We illustrate three session characteristics of three users on two types of devices. Each pair of colored lines (bars) correspond to one characteristic of one user's behaviors on a phone vs. a tablet. They present high-level patterns regarding user behaviors across devices.

How a user swipes and clicks may bear unique biometric information. It is desirable to include the information for user recognition.

To support normal usage of a mobile device, when we touch the screen of a device, dedicated hardware automatically generates data and reports them to the operating system as raw events. The system or the app further processes the raw data to understand user gestures and responds with corresponding functionality. Taking Android as an example, the raw event records measurements of *position, size, pressure* and *time-stamp* of a screen touch. Position is measured in terms of pixels along two dimensions of the device. It is fine grained. Size and pressure information are less accurate. Especially for pressure, some devices do not have dedicated sensor for this measurement. They thus report constant value or simulate the value from size measurement. We need to preprocess the data based on domain knowledge and observations to calibrate inherent differences across devices introduced by hardware.

In our app, there are icons for users to click on. Since their positions are fixed, their corresponding click measurements have negligible variance on position, and are thus less favorable as features. For articles, users scroll the list to find articles of interest, and click on the titles. In this sense, click positions have more variance and entropy for different users. As shown in Figure 6(a), users give different click positions. User 1's clicks are

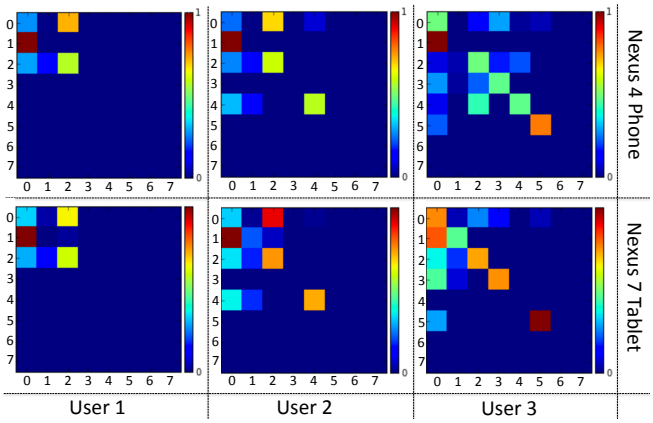
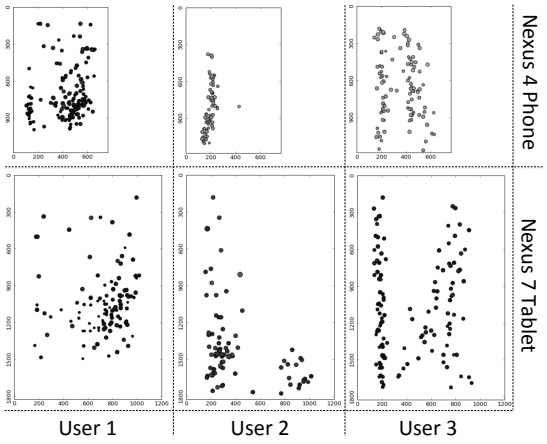
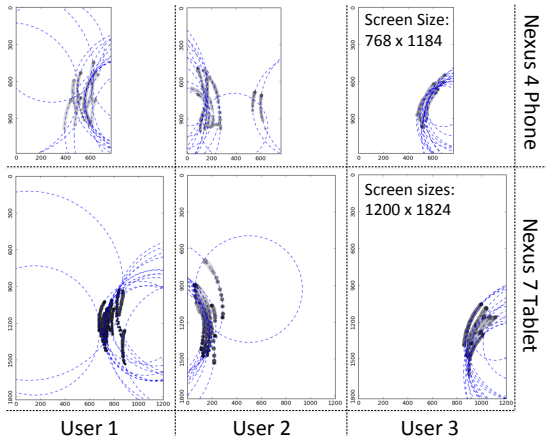


Fig. 5. To give a straightforward comparison of clickstreams, we visualize transition matrices as heat maps. The numbers 0-7 represent locations within the app such as top articles, new articles and bookmarks.



(a) A tap has position on screen, size and pressure. Each point in the above scatter plots represents a tap. Its size visualizes the size of the corresponding tap. The pressure is visualized as darkness. For each user on each device, we randomly and uniformly sample 100 clicks to illustrate their click preference.



(b) A swipe is a series of taps. In the figure, for each user on each device, we randomly and uniformly sample 15 swipes for illustration. We also fit a circle to each swipe to show its radius and direction.

Fig. 6. We depict users' taps on screen when clicking on article titles and swipes when scrolling the article list.

more spread on the screen. User 2 might be left-handed due to the imbalance to the left side of the screen. User 3 is equal between left and right.

Users swipe on the screen to browse articles. A swipe is a sequence of taps. Each tap has its position, size, pressure and time-stamp. We also fit a circle to each swipe to show its radius and direction. Figure 6(b) shows that swipes can be an even stronger indication of handedness; user 2 is most likely left-handed while users 1 and

3 are opposite. Despite that, user 3's swipes have smaller radius and are more concentrated than user 1, which adds further biometric features.

Click and swipe features are relatively sensitive to hardware differences that introduce inherent dissimilarity of measurements across devices. Instead of leaving it fully to our learning model to automatically compensate for the difference, we calibrate some difference beforehand using heuristics and domain knowledge. To give an example, for the pressure and size data, sensors on different devices have distinct sensitivity and measurement reference. With the tap data of the same users on different devices, we can learn the relative difference across devices and normalize the data in terms of mean and variance. For the position of clicks, we create relative values in addition to the deterministic values to facilitate generation of features that focus on relative positions.

To generate click and swipe features, we compute statistics in time and frequency domains. Table 1 summarizes the features.

4.2.4 Feature Summary. We extract both numerical and categorical features from aforementioned domains and more, giving a total of 173 features. To give a summary, we compute time domain features such as mean, deviation and frequency domain features including FFT values. Moreover, we include features unique to our study. For example, we divide the screen into six areas and compute percentage of clicks falling into each area. We employ the transition matrix of clickstreams to take into account app-specific behavioral patterns. We also add a feature to indicate if two devices belong to the same type or model so that our algorithm can compensate for inherent differences across devices.

4.3 Recognition Procedure

The multi-device case is formalized as a set partition problem. The recognition procedure comprises 1) *coarse-grained grouping*, and 2) *fine-grained pairing*.

4.3.1 Coarse-Grained Grouping By Overlaps. An app may have millions of installations. We do not directly apply supervised algorithms to all devices at that scale. A wiser way is to first break them down into small groups using rule-based heuristic methods. A common criteria for coarse-grained grouping is *location proximity*. The underlying intuition lays in the fact that devices of the same users are likely to co-locate with each other during certain time periods (it is theoretically possible for some users that their devices never appear together, in which case our approach cannot provide added-value for those devices). The co-location can be geographic or in terms of network address. Therefore, overlaps of two devices at physical locations or network segments enable the grouping which can significantly reduce the search space for devices potentially owned by the same users.

We admittedly have neither a theoretical justification for location-based grouping (which itself can be a serious study similar to the famous six degrees of separation [40]) nor an experimental study on any large scale dataset from industry, as they have legal concerns about publishing relevant results. We have only attempted to survey related literature to obtain understandings about this step. We have investigated papers about location-based user identification [5, 6, 20]. Locations can serve as quasi-identifier of users. With sufficient location information, we are able to uniquely identify a user. Mobile devices co-locate with their users. In fact, a main method to measure user locations is through their devices. In this sense, we are able to identify devices with locations. On the another hand, we survey papers on locations, geo-clustering and social relations [2, 10]. The finding is locations and social relations interact with each other. Social relations tend to imply location proximity. Hence, devices of the same users or socially close users bear similar proximity, which becomes the foundation for location-based grouping of devices.

An app can obtain location information of devices through GPS, WiFi SSID and IP address. The heuristic criteria to pair devices together is that they are frequently within physical proximity, connect to the same WiFi access points, or appear on the same IP segments for certain amount of time.

ID	Feature Description
fc_1, fc_2	The mean of click position in vertical and horizontal directions.
fc_3	The mean of click pressure.
fc_4	The mean of click size.
fc_5, fc_6	The variance of click position in vertical and horizontal directions.
fc_7, fc_8	Ratio of vertical clicks or horizontal clicks to all clicks.
$fc_9, \cdots, fc_{28}$	The index (frequency) of the 10 highest FFT value of click data.
$fc_{29}, fc_{33}, fc_{37}$	The max, min and median values of click position in vertical direction.
$fc_{30}, fc_{34}, fc_{38}$	The max, min and median values of click position in horizontal direction.
$fc_{31}, fc_{35}, fc_{39}$	The max, min and median values of click pressure.
$fc_{32}, fc_{36}, fc_{40}$	The max, min and median values of click size.
fs_1, fs_2	The mean of swipe trace in vertical and horizontal directions.
fs_3	The mean of pressure in swipes.
fs_4	The mean of touch area of taps in swipes.
fs_5, fs_6	The variance of swipe trace in vertical and horizontal directions.
fs_7, fs_8	Ratio of vertical swipe or horizontal swipe actions among all swipe actions.
$fs_9, \cdots, fs_{28}$	The index (frequency) of the 10 highest FFT value of swipe data.
$fs_{29}, fs_{33}, fs_{37}$	The max, min and median of swipe trace in vertical direction
$fs_{30}, fs_{34}, fs_{38}$	The max, min and median of swipe trace in horizontal direction
$fs_{31}, fs_{35}, fs_{39}$	The max, min and median of swipe pressure
$fs_{32}, fs_{36}, fs_{40}$	The max, min and median of swipe touch area
fs_{41}	The angle of moving during swiping
$fs_{42}, fs_{46}, fs_{50}, fs_{54}$	The mean, max, min and median of velocity during swiping in vertical direction
$fs_{43}, fs_{47}, fs_{51}, fs_{55}$	The mean, max, min and median of velocity during swiping in horizontal direction
$fs_{44}, fs_{48}, fs_{52}, fs_{56}$	The mean, max, min and median speeds of swipe pressure change
$fs_{45}, fs_{49}, fs_{53}, fs_{57}$	The mean, max, min and median speeds of swipe touch area change
fs_{58}	The acceleration of moving during swiping in vertical direction
fs_{59}	The acceleration of moving during swiping in horizontal direction
fs_{60}	The mean of swipe radius
fs_{61}	The mean of swipe direction at the start point
fs_{62}	The mean of swipe direction at the stop point
fs_{63}	The mean of swipe latency

Table 1. We summarize our features engineered from user clicks and swipes. The features include common statistics and frequency domain components. They capture characteristics of a user interacting with our app.

4.3.2 Fine-Grained Pairing By Usage Patterns. The coarse-grained grouping is straightforward yet effective in reducing the search space. After grouping, a user's device is likely to be mixed with devices of family, colleagues and friends since the social relations typically imply physical proximity of people and their devices as well. To further identify devices of the same owners, we leverage distinctions in usage patterns of different users. Since it is formulated as a set partitioning problem, the k-means algorithm might qualify as a candidate. However, it is not suitable for this problem due to two main reasons: 1) k-means requires a moderate sample size and does not perform well with sparse data [34] while our group size is well under 100 and the graph is sparse, and 2) k-means under-performs with high-dimensional data [45] while our features are of dimension 139. To tackle the challenges, we propose a fine-grained pairing method of two steps: *pairwise classification* and *refinement*.

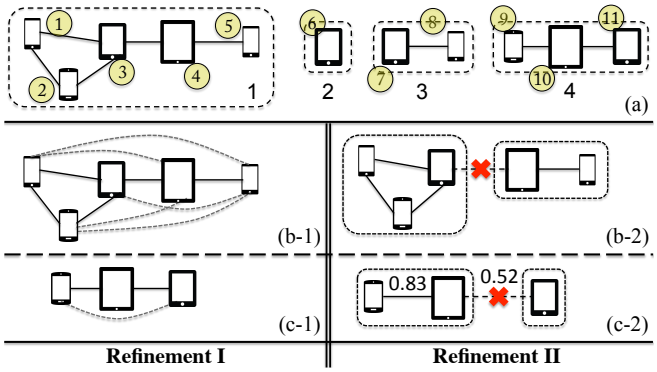


Fig. 7. We illustrate examples of refinement I and II after pairwise classification, which constructs fully-connected subsets by adding or removing certain edges.

After feature extraction, each device d_i has its feature vector fv_i. For any pair of devices $\langle d_i, d_j \rangle (i \neq j)$, we construct a feature vector fv_{ij} which measures the similarity between behaviors on the two devices. The pairwise feature vector fv_{ij} is defined as the element-wise squared vector difference $\|fv_i - fv_j\|^2$, which is widely used as a distance and similarity metric. The resulting feature vector is also of dimension 139. Additionally, we add an extra field to indicate whether the two devices are of the same type in terms of phone or tablet so that learning algorithms can take into account inherent difference between devices. With this pairwise setup, the first step is to determine the $(0, 1)$-label for each pair $\langle d_i, d_j \rangle$ to indicate whether they belong to the same user.

XRec employs random forest to classify device pairs. Random forest is an ensemble learning method that constructs many decision trees and outputs the mean prediction of individual trees. It is able to address non-linear features and inherently performs feature selection. Random forest achieves good robustness over single classification approaches [9, 14]. Moreover, we use regression to yield a mean prediction between 0 and 1, which can serve as a confidence score. Then we introduce a threshold to generate $(0, 1)$-labels.

The second step after pairwise classification is refinement. With each device d_i as a vertex in a graph $\mathcal{G}$, there exists an edge between two devices $\langle d_i, d_j \rangle$ if the predicted label for the pair is 1. We can populate all potential edges based on classification results. The graph of devices is thus partitioned into connected subgraphs. A graph is connected if there is a path from any vertex to any other vertex in this graph. Figure 7(a) illustrates a sample output of pairwise classification. Each subset is expected to by *fully connected* which does not hold for subset 1 and 4. The full connectivity requirement comes from the fact that all devices in a subset are supposed to belong to one user. Conflict arises when dev_1 and dev_2, dev_2 and dev_3 belong to the same user while dev_1 and dev_3 do not. Our refinement algorithm updates edges so that all subsets fulfill the fully connectivity requirement. We consider two approaches: 1) *refinement I* adds edges to connect nodes in a partially connected subset, and 2) *refinement II* removes edges to partition a subset into several fully connected subsets. Sub-figures (b-1) and (c-1) illustrate two examples of the refinement I which is straightforward. The refinement II keeps fully connected subsets and remove extra edges. Sub-figure (b-2) presents a simple example: removing one edge partitions the set into two fully connected subsets. A tricky scenario is depicted in (c-2) where we can remove either edge to succeed. In certain cases, we want to delete the edge with lower confidence score.

Refinement II alternatively iterates through two steps: 1) finding the maximum clique in the current graph, and 2) removing the maximum clique and all associated edges from the graph. In the presence of multiple maximum cliques, we choose one such that the associated edges to be removed have lowest confidence scores. To find the maximum clique, we use existing maximum clique algorithms [8]. Readers may refer to Algorithm 1 for details.

Algorithm 1: Refinement Algorithm II — Reduction

Input: Adjacency matrix $\mathcal{A}_{n\times n}$, confidence score $\mathcal{S}_{n\times n}$
Output: New adjacency matrix $\mathcal{A}'_{n\times n}$
Procedure Refinement
$\mathcal{A}'_{n\times n} = \mathcal{A}_{n\times n}$;
while $\mathcal{A}_{n\times n} \neq [0]$ **do**
    $cliques$ = FindMaximumClique($\mathcal{A}_{n\times n}$);
    if $cliques.size() < 2$ **then**
        break;
    end
    $clique = \emptyset$;
    $edgesToRemove = \emptyset$;
    $minScore = +\infty$;
    for c ***in*** $cliques$ **do**
        $victims$ = FindEdgesToRemove($\mathcal{A}_{n\times n}, c$);
        $scores$ = GetScores($\mathcal{S}_{n\times n}, victims$);
        if $scores < minScore$ **then**
            $edgesToRemove = victims$;
            $minScore = scores$;
            $clique = c$;
        end
    end
    RemoveEdges($\mathcal{A}'_{n\times n}, edgesToRemove$);
    RemoveClique($\mathcal{A}_{n\times n}, clique$);
end
return $\mathcal{A}'_{n\times n}$;

Refinement II uses binary classification to achieve satisfying recognition performance. In the binary classification phase, we can tune parameters to intentionally increase recall (namely, predict more edges in the graph). Then, in the refinement stage, the algorithm removes edges in accordance with the topological constraint that each clique should be fully connected. The topological constraint provides extra information in addition to behavioral patterns on different devices. Most of the removed edges are false positives. Later in Section 6.2.3, we present an illustrative example and experimental results to show the effect of refinement.

5 EXPERIMENT SETUP

In this section, we present our experiment setup. The experiment concentrates on fine-grained pairing using real usage data, obtained under IRB approval.

5.1 Data Collection

In this cross-device recognition study, each subject in our study uses the experimental app on four types of devices: Nexus S, Nexus 4, Nexus 7-2012 and Nexus 7-2013. Each subject uses the app on two devices of each type, yielding eight sets of usage data per subject. Though it is rare for a real user to own eight devices, with this setup we do not actually mean for them to have eight devices. We can opt to use part of the data to form different scenarios. Four pairs of different devices allow us to study usage behaviors of each subject on 1) identical devices, 2) same type but different models (Nexus S phone *vs.* Nexus 4 phone), and 3) different types of devices (Nexus 4

phone *vs.* Nexus 7 tablet). Also, with eight sets of usage data per subject, we have the flexibility of evaluating the problem for cases of various numbers of devices per user from one to eight. Moreover, with this setup we can test the performance of our approach in extreme case and also obtain insights of user behaviors on different types of devices.

In addition to usage behaviors of each subject across eight devices, we also need to study the variation of behaviors on the same devices by different subjects. To this end, we recruit twenty subjects from different demographic groups to participate in the data collection, giving essentially 160 sets of usage data in terms of the $\langle user, device \rangle$ pair. With the data, we are able to study inter-user, intra-user, inter-device and intra-device behavioral similarity or difference. Moreover, per our discussion in Section 4.3.1, the coarse-grained grouping can significantly narrow down the cross-device recognition to within relatively small groups. From the 160 sets of usage data, we can effectively construct such groups for studying fine-grained pairing within the groups. That said, we admittedly note that it is favorable to include more devices and participants into this user study. However, our effort is limited by the constraints of time and hardware. We do not have dozens of mobile devices for simultaneous data collection from many participants, and collecting data on eight devices for even one user is already time-demanding. In this paper, we hope to deliver proof-of-concept results on this new cross-device recognition problem, which we believe can be fulfilled with the 160 sets of usage data.

After we distribute devices to a subject for data collection, we do not give specific instructions to use the experimental app other than a basic introduction of the app. They use the app in accordance to their own preference and habit. In this fashion, we hope the subjects exhibit their own patterns in the usage that are not biased by any particular recipe. Each subject uses the experimental app on eight devices. This does not mean they need to view the same content from the app on all devices. The subjects switch between devices in their usage. For the twenty subjects, we have collect usage data of effectively two to ten weeks, including hundreds to thousands of clicks and swipes.

Last, due to limited resources, the device models we use are only a small portion of all models on market. The four device types in our experiment consist of phones and tablets of different models, sizes and hardware. Although they are not completely representative of all devices, they present differences we hope to have and deliver meaningful results.

5.2 Experimental Data Generation

Without access to large-scale data that a commercial app provider would collect, we have attempted to approximate various scenarios. We collected usage data of real users using our app on real devices. However, we are not able to attract a large number of users to study the coarse-grained grouping step. Therefore, we simulate probable grouping outputs using the 160 sets of usage data. There are three primary parameters describing a group: 1) total number of devices of the group, 2) number of devices per user, which determines the number of underlying users given the total number of devices, and 3) diversity of devices which defines the number of different types of devices. With respect to the set partitioning problem, the first parameter specifies the size of the set to be partitioned. The second parameter tells the number of clusters. The third parameter describes diversity which affects the similarity and difference between behaviors on devices within a group.

Our data allow us the flexibility to simulate various groups of different parameter settings. With 160 sets of $\langle user, device \rangle$ data, we can simulate a group of up to 160 devices with its correct partition being twenty subsets each of eight devices, which might be rare but can test some properties about our approach such as scalability. The size will be smaller considering the withheld training data in our supervised approach.

We will conduct several experiments in the next section. Without otherwise stated, we use 80 devices from ten users as training data, and the remaining for testing. Additionally, the results are averaged over ten different sets of training-test split. Some other details about the setup will be presented along with results in the next section. The training phase essentially models similarities or differences of behaviors on 80 devices, which are the

distances between behaviors represented as feature vectors instead of isolated behaviors on individual devices. In training, we include all eight devices of ten users to obtain sufficient positive and negative instances of device pairs. A positive instance refers to the case that two devices belong to the same user, while a negative instance denotes the opposite. Having two devices of 100 users can also help achieve the goal of having sufficient positive and negative instances. Unfortunately, we are not able to perform a study at that scale. Though it might be difficult to find real users with eight devices, the setting does not undermine practical implications of this study. Having eight device from each user gives no advantages to modeling. It is more about having enough positive instances with fewer users under our experimental settings. Additionally, we will evaluate XRec against different training settings where we have fewer training instances.

6 PERFORMANCE EVALUATION

In this section, we evaluate our approach in terms of its recognition performance. We aim to provide insights by investigating the following questions.

- What is the overall performance of XRec?
- How does the group size affect recognition results?
- How does the recognition performance change with the number of devices per user on average?
- How well can our approach perform with a mixture of multiple different device types?
- What is the generalization capability in terms of users, i.e. can XRec generalize from a few users to many others?
- What is the generalization capability in terms of devices, i.e. can XRec generalize from a few devices to other devices?

6.1 Performance Metrics

Recall from previous discussion, the user recognition problem can be formalized as *set partition*. To evaluate XRec, we need to compute the distance between the estimated partition $\hat{\rho}$ and the correct partition ρ. Denoeud et al. [12] compared several distance metrics between set partitions such as the Rand index, the Jaccard index, the corrected Rand index, the Wallace index and the normalized Lerman index. We employ two widely used metrics for partitioning evaluation: the Jaccard and Rand index.

To compute the *Jaccard index* between $\hat{\rho}$ and ρ, we need to evaluate on all pairs of elements in the set. For any pair $\langle x, y \rangle$, they can be joined or separated in each partition: joined means they are labeled as 1 while separated means the opposite. We denote as r the number of pairs simultaneously joined in both partition, s the number of pairs simultaneously separated, u the number of pairs joined in $\hat{\rho}$ and separated in ρ, and v the number of pairs separated in $\hat{\rho}$ and joined in ρ. Then the Jaccard index between the two partitions is defined as:

$$\mathbb{J}(\rho, \hat{\rho}) = \frac{r}{r + u + v}.$$

The Jaccard index omits s since its inventor believes a partition is typically interpreted as joining elements. In our evaluation, we may also want to reward correct separations. The *Rand index* gives equal emphasis on simultaneously joined or separated pairs. We also assess our approach using this index to offer another angle of perspective. The Rand index is defined as:

$$\mathbb{R}(\rho, \hat{\rho}) = \frac{r + s}{r + s + u + v}.$$

The evaluation metrics on group partitioning are in terms of pairs within the group. From the perspective of pairwise binary classification, the aforementioned r, s, u, v are respectively true positive (tp), true negative (tn), false positive (fp) and false negative (fn). In this sense, the Rand index is essentially the *accuracy*. To help

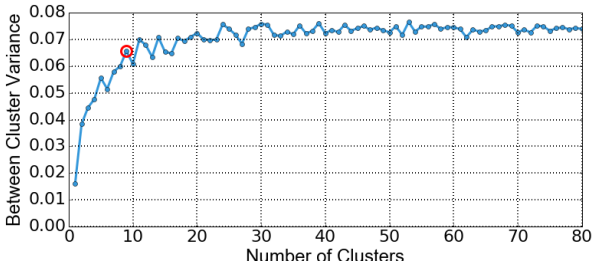


Fig. 8. We illustrate the elbow method used in the k-means algorithm when the number of clusters is not known. It examines the percentage of variance explained and identifies the 'elbow' point indicated by the red circle.

understand the effect of our refinement algorithm, we also use metrics defined as follows:

$$precision = \frac{r}{r+u} = \frac{tp}{tp+fp},\ recall = \frac{r}{r+v} = \frac{tp}{tp+fn}.$$

6.2 Recognition Performance At A Glance

First, we evaluate on all test data of 80 devices from ten users. Even though a group of 80 devices might be rare, we hope the result can give an overall impression on the performance, and also demonstrate the scalability of the algorithm. We also compare the performance with several baseline approaches. Moreover, we present analytical results to highlight the effect of the refinement algorithm.

6.2.1 Baseline Approaches. Our approach is compared to three baseline approaches including: 1) a naïve baseline method (`Naïve`); 2) k-means algorithm (`Kmeans`); and 3) a heuristic method (`Heuristic`).

The `Naïve` baseline randomly guesses (0,1)-labels for all pairs during the binary classification phase and employs our refinement algorithm I and II. The `Kmeans` approach incorporates the elbow method [37], which is used to determine the number of clusters in k-means. As shown in Figure 8, the method plots the percentage of variance explained by the clusters against the number of clusters. The 'elbow' point is indicated by the red circle. The `Heuristic` method trains a distance threshold on labeled data that maximizes the Jaccard value. Then the distance is used to link any pair within the threshold during the binary classification. Refinement methods I and II are applied subsequently.

Both the `Naïve` and `Heuristic` methods require refinement since they first make a binary prediction of each pair. The `Kmeans` approach, on the other hand, directly partitions the device set and requires no refinement.

6.2.2 Performance Comparison. As shown in Figure 9(a), XRec with refinement II achieves the highest performance by Jaccard index, Rand index and precision. For 80 devices and thus $\binom{80}{2} = 3160$ pairs, XRec achieves 70% Jaccard value, 98% Rand value and precision, and 70% recall. XRec with refinement I obtains high recall value, but loses precision. Refinement I causes low performance for `Naïve`, `Heuristic` and XRec since it blindly increases recall by aggressively adding positive instances, many of which are unfortunately false. On another hand, refinement II leverages a topology constraint and makes appropriate adjustments to the binary classification results. We will present experimental results shortly to further reveal its mechanism.

As also shown in the figure, XRec performs significantly better than all baseline results. Overall, the Rand value is satisfying since there are many more negative instances than positive ones and it is relatively easy to correctly predict the negatives. However, for the baseline approaches that employ refinement I, the Rand values decrease significantly since refinement I predicts many negative instances as positive. The recall values for some baseline approaches are satisfying. The high recall is due to the fact that these methods make sufficient amount

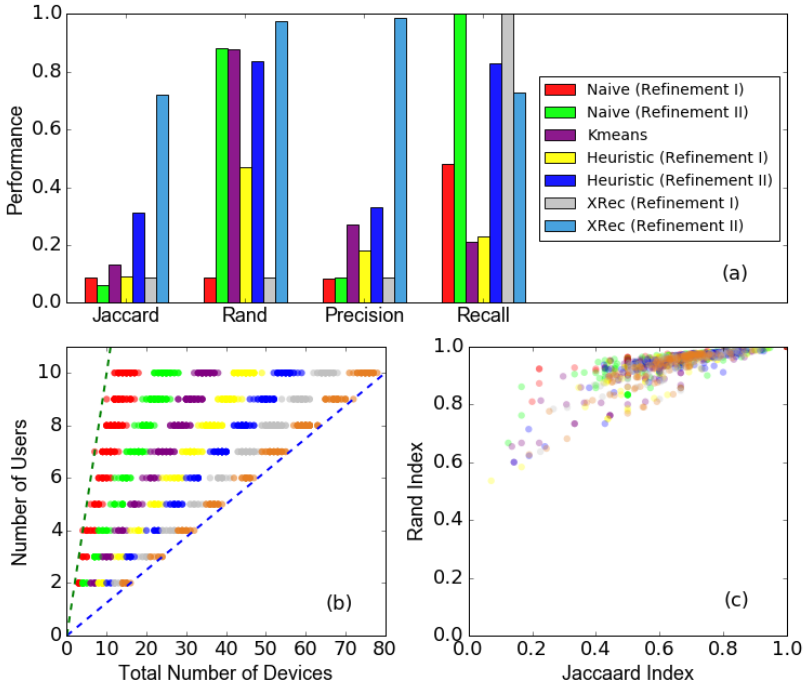


Fig. 9. In figure (a), we contrast XRec with baseline approaches. Refinement I and II are also compared. In figure (b) and (c), we further illustrate its performance for different group outputs from coarse-grained grouping.

of positive predictions so that the true positives are predicted. All baseline methods underperform in terms of Jaccard and precision values.

The `Naïve` method makes random guesses and gives a lower bound for the performance. Its precision and Jaccard values are low regardless of the refinement methods employed. `Kmeans` slightly outperforms the `Naïve` method. The `Kmeans` method, as a clustering method in nature, makes use of topological characteristics of the group of devices. However, without knowledge of the underlying number of clusters, the uncertainty compromises its performance. Additionally, the high dimensionality of the feature space makes clustering even more difficult. The `Heuristic` with refinement II achieves about 30% Jaccard value, outperforming other baseline approaches with a Jaccard value below 15%. This method learns a distance threshold to maximize the Jaccard value. The learning process takes into account behavioral similarity between devices and topological characteristics of groups. The excessive amount of information incorporated in the learning process creates sparsity. When the learned model is applied to test data, the performance suffers from the lack of generalizability because the test data might exhibit dissimilar behavioral and topological patterns.

Second, we want to evaluate XRec on test data of diverse formations representing various group scenarios. To this end, we randomly sample from the whole test data to construct a variety of different groups. A group is defined by the three parameters as discussed before: 1) total number of devices, 2) number of underlying users, and 3) number of different types of devices. To construct different types of groups, we first sample two to ten users. For each user, we sample one to eight devices. In total, we obtain 2000 sets of test data. Figure 9(b) shows two of the three parameters for the 2000 groups. The two dotted lines define possible area for the two parameters as each user has one to eight devices. For each group, we depict its Jaccard and Rand indexes obtained

in Figure 9(c). As shown in Figure 9(c), points concentrate on upper-right corner. The variance of performance comes from not only the difference in group formation but also specific users included in certain groups. If some users in a group have very similar behavioral patterns, the performance will be undermined since our approach relies on behavioral distinctions. In this case, the app simply loses the potential added-value brought by XRec on these users if they happen to be in the same group. On average for all randomly sampled 2000 groups, XRec (refinement II) achieves 91% Rand value and 61% Jaccard value. Since XRec relies on behavioral patterns of users and the topological constraint, its performance hinges on various factors. For some specific group scenarios, the performance is far below the average level. It might be caused by various reasons such as: 1) some users in the same group exhibit similar behaviors, 2) the group is small and each misclassification thus incurs high performance loss. To further improve the recognition performance, we may want to introduce new features that capture other aspects of user behaviors. For instance, we can possibly use accelerometer data to reveal user's holding posture. This potential improvement comes with the price of extra overhead since accelerometer has a relatively high sampling rate. Moreover, it requires extra permission to access the data.

The added value brought by XRec can enhance advertising, recommendation and user experience. The working mechanism is similar to cookie-based online tracking. Web cookies track users across websites while XRec tracks users across mobile devices. The primary goal for both is to link the current user to his previous activities. In this sense, XRec works in parallel with the advertising and recommendation systems. XRec makes predictions about the linkage among users on different devices. Then the content viewed by the user on other devices also indicate his interest and can be leveraged for optimization on the current device. The XRec outcome is fed to recommendation algorithms so that they can make use of the history content viewed by the user to recommend new content the user might be interested in. The same procedure applies to advertising. The past trace exposes user interest and there is higher chance that related advertisements being clicked. XRec also offers opportunity for enhancing user experience with the app. For example, a news article left open on another device can be displayed on the current device in case the user may want to continue reading that article. Though the experience optimization process is app specific and requires careful design and implementation, the information relating to user identity is generally useful. XRec aims to serve downstream services that favor a notion of user identity to better fulfill their goals.

In this experiment, XRec achieves 70% recall and 98% precision. The recall is a measurement of the percentage of true positives that get predicted. If we predict all instances as positive, all true positives are predicted as positives. The recall value is 100%. However, all negatives are also predicted as positives, incurring many errors. The precision offers another perspective. It measures the percentage of true positives among all predicted positives. Hence, if we do not make any positive predictions, we do not even risk making mistakes. The precision value will be high. In this sense, there is a trade-off between recall and precision. With 70% recall, it means 70% of device pairs that belong to the same users are predicted as positives. They are the true positives. The remaining 30% are hence false negatives. From the perspective of precision, 98% of predicated positive pairs prove to belong to the same users. Only 2% of them are actually owned by different users. This is important to conservative applications that tend to predict less positives yet predict correctly. The four metrics used in this study measure the partitioning performance, which cannot be directly translated into the performance gain of downstream applications based on XRec. For example, an advertising application can leverage XRec outcomes to deliver more relevant advertisements. The improvement on click-through-rate will depend on several factors such as ads quality and relevance, user intent and engagement. Even though it is difficult to quantify the impact of XRec in this process without experimental study, a satisfying partitioning performance can contribute to the downstream application by providing this notion of identity across devices.

6.2.3 Effect of Refinement. We also reveal the working mechanism of the proposed classification-plus-refinement algorithm. The pairwise classification outputs confidence scores from 0 to 1. A boundary is chosen

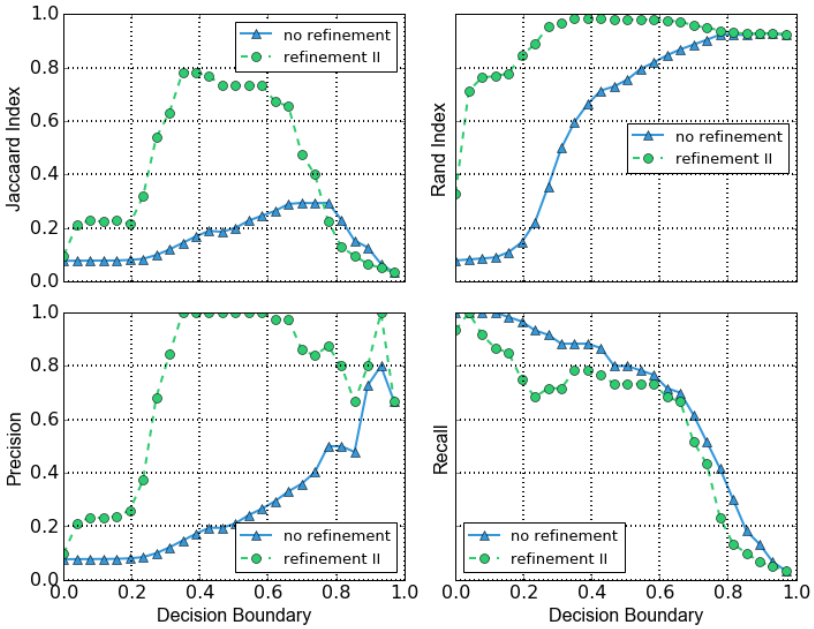


Fig. 10. We illustrate analytical results under different decision boundaries to demonstrate how refinement boosts recognition performance. The refinement after binary classification significantly enhances precision without sacrificing much recall.

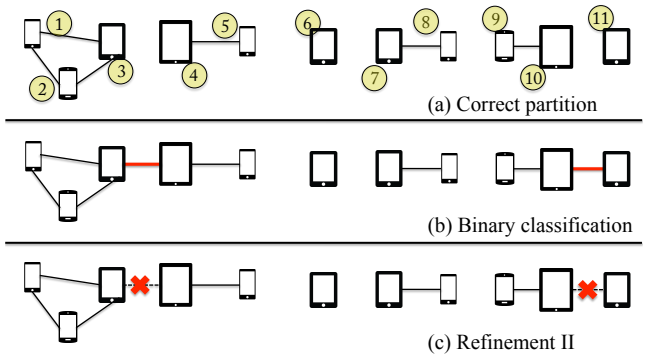


Fig. 11. We present an illustrative example to show how refinement can improve recognition performance. The binary classification excessively predicts associated pairs, which creates false positives (indicated as red lines) compared to the correct partition. The subsequent refinement procedure selectively removes edges, most of which are false positives.

to further produce (0,1)-labels indicating whether a pair of devices belong to the same user. Figure 10 shows the effect of the decision boundary on the performance of refinement II measured in four metrics. We use 80 devices as training data and the remaining 80 devices as test data. In terms of all four metrics except recall, we can observe that performance with refinement II is consistently better than the performance without refinement. Refinement II only slightly reduces the recall value by about 5%. By wisely removing positives as shown in

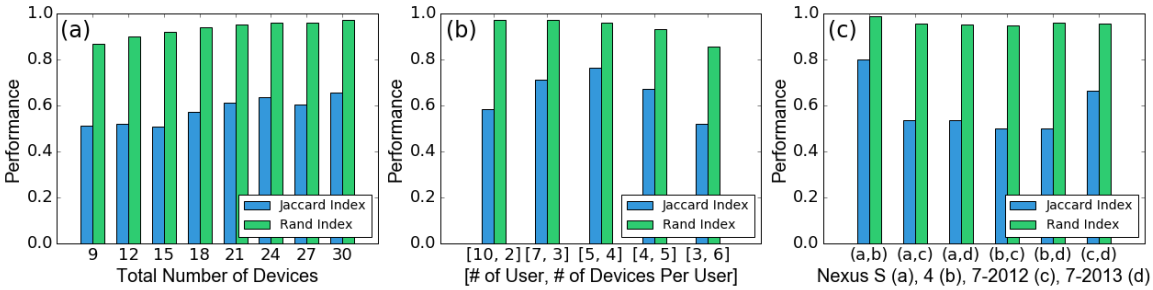


Fig. 12. We illustrate the performance of XRec against changes of three controlling factors that describe a group scenario.

Figure 7, refinement II boosts precision by 80% without sacrificing much recall. *Recall* evaluates if all true positives get predicted, while *precision* assesses if the predicted positives are true. The balance between recall and precision can be achieved by adjusting the decision boundary in binary classification. As shown in the figure, a suitable decision boundary value from 0.4 to 0.6 gives satisfactory results in our experiment.

Figure 11 presents an illustrative example how the refinement based on the topological constraint improves performance. Sub-figure (a) shows the correct partitioning of eleven devices. In the binary classification, we intentionally adjust the decision boundary to predict more positive instances, maintaining high recall value. However, aggressive prediction generates some false positives indicated as red lines in sub-figure (b). Subsequently, the refinement phase removes a substantial amount of false positives by enforcing the topological constraint (namely the fully-connection clique constraint) in the graph.

6.3 Three Controlling Parameters

As mentioned above, a group is described with three parameters. We study the effect of different group scenarios on performance. We employ XRec with refinement II, and use the same model trained in the above experiment.

6.3.1 Comparison of Different Group Sizes. Group size is the total number of devices in a group obtained from coarse-grained grouping. Depending on the strictness of the grouping method, physical proximity of devices, population density and other factors, the group size may vary. To generate test data for this problem, we fix the number of devices per user at three: a Nexus 4, a Nexus 7-2012 and a Nexus 7-2013, while the number of users changes from three to ten (the maximum number of users available in test data), giving a range of group size from 9 to 30. We generate nine sets of data per setting except for the number of users being ten, and compute the average performance value.

Figure 12(a) plots the results. For most settings, XRec can achieve about 55% Jaccard value and 90% Rand index. With the increase of devices, the performance is on a slight upward trend. An explanation is more devices introduce more positive instances which is captured by the Jaccard index. Moreover, the refinement can reduce false positives. The combined effect brings the values slightly higher. Intuitively, a smaller group size makes the partitioning task less challenging. However, it also depends on the dissimilarity of behaviors from different users. If users within a small group happen to share common behavioral patterns. XRec is likely to treat them as the same user. Moreover, when the group size is small, each misclassification can incur higher performance loss.

6.3.2 Number of Partitions. The number of partitions of a group is essentially the number of underlying users. To prepare test data, the group size is fixed at 20 devices. Varying the number of partitions from three to ten, we randomly generate several corresponding partitions. For instance, three implies partitions such as (8, 8, 4), (8, 7, 5).

Moreover, the three users are randomly chosen from all ten users. For each setting, we generate ten sets of test data and obtain average performance.

As shown in Figure 12(b), XRec achieves above 50% Jaccard value and 80% Rand value. The case of 5 users each with 4 devices gives best Jaccard value. Going either side decreases the value. Similarly, the reason might be the number of positive instances in each combination. The true positives can be well-presented by the Jaccard index. Moreover, the refinement in XRec relies on the topological constraint that every clique is fully connected. With 5 users and 4 devices each, the clique size and the number of cliques are medium. A graph with many small clusters is sparse. It is easier for XRec to output small cliques since they require much less mutually connected pairs. For example, an edge alone can generate a two-node clique. The performance is penalized if a small clique is actually incorrect. On another hand, a graph with a few large clusters is dense. In this type of graphs, to output a large clique, XRec needs to correctly predict all pairs within the clique. Missing edges will break the large clique into smaller ones, and incur performance loss. From this perspective, a medium number of cliques of medium size can potentially benefit the partitioning.

6.3.3 Different Combinations of Devices. Our experiment uses four device types. We are interested in possible variance between recognizing users across two phones and across phone-tablet. To prepare test data, we fix group size at twenty: ten users each of two device types.

Figure 12(c) shows the results for different combinations. Identifying users across Nexus S and 4 reaches 80% Jaccard value, while it is about 50% for the phone-tablet combinations, and 65% between the two tablets. The recognition across the same type of devices outperforms that across phone-tablet. User behaviors on the same type of devices tend to be similar. The screen size and hardware on different devices might cause subtle changes on some aspects of the behaviors. In this sense, XRec loses some discriminating information from those device-dependent features. The calibration during data preprocessing can help with the situation to some extent. Our recognition is probabilistic in nature. From this perspective, the recognition made for the same type of devices has higher chance to be correct. This observation can be taken into consideration by downstream services when they make use of the XRec outputs.

6.4 Different Settings for Training Data

In this section, we change our training setting to investigate how XRec generalizes its knowledge on users and devices. We also provide insight into how much training data are appropriate to achieve satisfactory performance. We only use a subset of the original training data. The test data consist of the remaining 80 devices.

6.4.1 Cross-User Generalization. For training, we choose two to ten users each with eight devices. We plot our result in Figure 13(a). XRec achieves above 60% Jaccard value and the trend is stable. Even with two users each of eight devices, XRec can capture behavioral differences among users across devices. It reduces the demand for extensive training samples. By the design of XRec, the input feature vector to the binary classification algorithm is the *difference* between behavioral patterns on two devices. The algorithm aims to capture the relative difference instead of absolute patterns, which lowers the importance of learning the patterns of any specific users. In this sense, a few users on multiple devices can offer sufficient instances for XRec to model behavioral differences across devices. This is a favorable advantage even though it is not difficult to recruit more users.

6.4.2 Cross-Device Generalization. We vary the number of devices per user from two to eight with ten users for training and 80 other devices for test. Figure 13(b) shows that the Jaccard value initially increases with the number of devices per user and then becomes stable. With two or three devices, XRec might fail to capture behavioral differences of users between enough devices of different types. As we use more devices per user for training, XRec observes more devices and can better model the differences. In this experiment, the test dataset contains all eight devices. It is desirable that XRec can observe instances from all devices in training. If the test

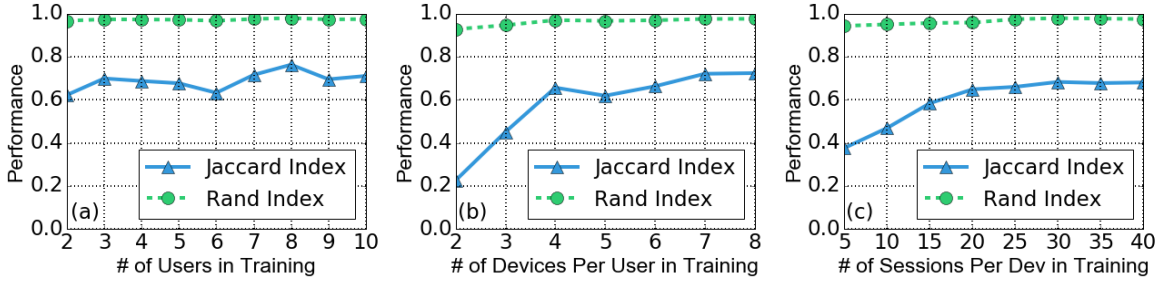


Fig. 13. We illustrate how different settings of training data affect recognition performance. In (a), the training data contain data from different numbers of distinct users. It aims to show whether the learned model can be applied to other users. In (b), the training data contain data from different numbers of devices. It attempts to show whether the model can be applied to other devices. In (c), the training data vary in length, showing the effect of the amount of training data on performance.

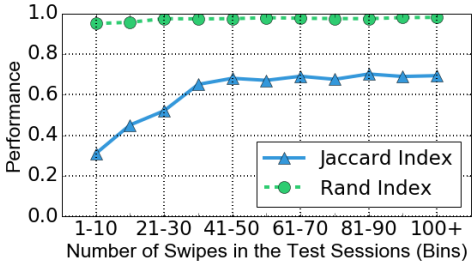


Fig. 14. We present the performance results when test sessions contain various numbers of swipes (an indicator of the effective length of a session). We observe an upward trend as more behavior data are available for modeling session users. The performance flattens out when extra data bring no additional entropy.

data and the training data come from different distributions, the trained model is likely to underperform for the test data. Hence, if we do not have all devices in the training data, we hope the devices at least are different in type so that the algorithm can incorporate this difference into the model. In real practice, XRec favors to include more types of devices into its training dataset so that the optimal performance can be achieved.

6.4.3 The Amount of Training Data. To study the effect of the amount of training data, we use 80 devices as training data and the remaining 80 devices for test. This setting is the same as the experiment shown in Figure 9. However, we vary the amount of training data per device, measured as the number of sessions. We randomly sample the sessions from a device's complete data.

Figure 13(c) shows the performance. The Rand index is constantly high since this measurement rewards correctly predicted negatives. The Jaccard value starts with about 40% with 5 sessions per device. It gradually increases to about 68% as the learning algorithm observes more data from each device. More data help the algorithm better model the behavioral patterns of app users. The increasing trend flattens out after about 20 sessions as extra data contribute no additional entropy for discrimination.

6.5 The Effect of Test Data Length

Performance concerns arise when a session with the app is short and there is insufficient data to model the behaviors of the current user. Hence, we are also interested in the effect of test data length. To set up the

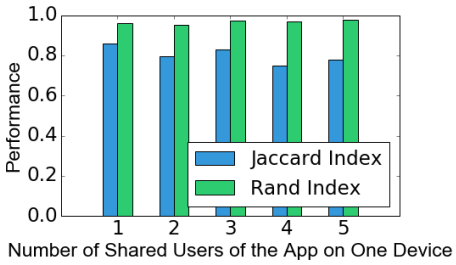


Fig. 15. We present the recognition performance on individual devices. We study the scenarios where the app has one to five users on each single device. XRec achives about 90% Rand index and above 70% Jaccard index.

experiment, we use 80 devices from ten users as training data. For the test data, we use data from the remaining 80 devices. Some sessions might not contain sufficient information for behavior modeling. The duration of a session is not a good metric since some long sessions are actually inactive and lack usage data. We observe that users generate more swipes than clicks in their use of this news reader app. We partition the sessions based on the number of swipes which indicates the *effective length* of the test data. The bins we use for the partition are $1-10, 11-20, 21-30, \cdots, 91-100, 100+$ swipes per session. In one experiment, each bin has 30 sessions. We report the averaged results over ten experiments in Figure 14.

The Rand index maintains high values above 90% even when the effective length is short. The Jaccard value starts from 30% when there are only less than 10 swipes in the sessions. The low engagement of users renders behavior modeling less accurate. Nevertheless, the data can still indicate some patterns of users such as handedness, swipe speed, position and so forth. As there are more swipes, the Jaccard index climbs to about 67% and stays on that level. More data provide more entropy and improve the recognition performance. The trend eventually converges as extra data bring no extra entropy for our algorithm.

There could be many inactive sessions in which users have limited interactions with the app. The lack of behavior data compromises the accuracy in user modeling and subsequently the recognition performance. From our data, 76% of sessions have more than 20 swipes. The small size of screens of mobile devices makes list layout a common design choice of many apps. Swipe is hence a frequent gesture for users to interact with the apps.

6.6 Session Segmentation on Individual Devices

While this paper concentrates on the problem of *cross-device* recognition, we also want to show experimental results of session segmentation on individual devices. Typically, device sharing happens on family-held devices. Moreover, the number of users frequently using the same app on the shared device is even less since they might have different interests and preferences. In our experiment, we study the session segmentation on all eight devices each with one to five users. The training data remain the same using ten users' data on eight devices. The remaining ten users' data are used as test data. For each of the eight devices, we simulate different scenarios where the app has one to five users on one device. For each device, we partition 50 sessions. For each number of users, we generate 100 group scenarios. For instance, for one device with three users, we randomly sample 50 sessions from three users and evaluate the performance on the data. We repeat this procedure 100 times and obtain the average performance for that device under the scenario of three users. The same procedure applies to other all devices with different numbers of users.

Figure 15 shows the average performance on eight devices measured in Jaccard and Rand indices. The Rand index exceeds 90% regardless of number of the shared users. The Jaccard index achieves above 70% for all cases. The performance on individual devices surpasses that in the cross-device recognition. A user's behavior tends

to be more consistent on the same device, which makes pattern recognition less challenging in this scenario. Moreover, the number of shared users on one device is smaller compared to the total number of users across multiple devices. The data are more concentrated than distributed. The combined effect improves the overall performance on single devices.

7 CONCLUSION

We present the design, implementation and evaluation of XRec for cross-device user recognition. Distinct from existing research work that investigates user identification or authentication by the operating system on an individual device, XRec deals with tracking users across multiple mobile devices by an app provider's back-end. Moreover, previous work typically concentrates on binary identification of owner or non-owner of a single device while our problem inherently demands multi-class recognition of multiple users across multiple devices. To tackle challenges present in this problem, we propose to use coarse-grained grouping based on location proximity plus fine-grained pairing based on usage behaviors. Algorithmically, XRec leverages pairwise classification plus refinement to solve the particular partitioning problem required for fine-grained pairing in this context. We evaluate XRec on 160 sets of usage data on eight devices of four types. For different group scenarios, XRec achieves an average Rand value of 91% and Jaccard value of 61%. These results offer confidence that user behaviors provide entropy for recognition even across device types and form factors. XRec offers the opportunity to optimize many downstream services across devices such as recommendation, advertising and user experience enhancement. XRec provides a notion of identity to these services so that they can track down users' activities across multiple devices. The history activities on other devices also indicate a user's interest and preference. XRec enables the mining of data at the user level instead of at the device level, enriching information available to other services for better performance.

REFERENCES

[1] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. 2014. The web never forgets: Persistent tracking mechanisms in the wild. In *CCS*. 674–689.

[2] L. Backstrom, E. Sun, and C. Marlow. 2010. Find me if you can: improving geographical prediction with social and spatial proximity. In *WWW*. 61–70.

[3] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. 2009. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference (IMC '09)*. 280–293.

[4] Salil P Banerjee and Damon L Woodard. 2012. Biometric authentication and identification using keystroke dynamics: A survey. *Journal of Pattern Recognition Research* 7, 1 (2012), 116–139.

[5] A. Beresford and F. Stajano. 2004. Mix zones: User privacy in location-aware services. (2004).

[6] C. Bettini, X. Wang, and S. Jajodia. 2005. Protecting privacy against location-based personal identification. In *Secure data management*. 185–199.

[7] Cheng Bo, Lan Zhang, Xiang-Yang Li, Qiuyuan Huang, and Yu Wang. 2013. Silentsense: silent user identification via touch and movement behavioral biometrics. In *Proceedings of the 19th annual international conference on Mobile computing & networking*. ACM, 187–190.

[8] I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo. 1999. The maximum clique problem. In *Handbook of combinatorial optimization*. 1–74.

[9] L. Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[10] J. Cranshaw, E. Toch, J. Hong, A. Kittur, and N. Sadeh. 2010. Bridging the gap between physical location and online social networks. In *Ubicomp*. 119–128.

[11] A. De Luca, A. Hang, F. Brudy, C. Lindner, and H. Hussmann. 2012. Touch me once and i know it's you!: implicit authentication based on touch screen patterns. In *HCI*. 987–996.

[12] L. Denœud and A. Guénoche. 2006. Comparison of distance indices between partitions. In *Data Science and Classification*. 21–28.

[13] S. Dey, N. Roy, W. Xu, R. Choudhury, and S. Nelakuditi. 2014. Accelprint: Imperfections of accelerometers make smartphones trackable. In *NDSS*.

[14] T. Dietterich. 2000. Ensemble methods in machine learning. In *Multiple classifier systems*. 1–15.

[15] B. Draffin, J. Zhu, and J. Zhang. 2014. Keysens: passive user authentication through micro-behavior modeling of soft keyboard interaction. In *Mobile Computing, Applications, and Services*. 184–201.
[16] Econsultancy. 2014. More than 40% of online adults are multi-device users: stats. (2014). https://econsultancy.com/blog/64464-more-than-40-of-online-adults-are-multi-device-users-stats/.
[17] T. Feng, Z. Liu, K. Kwon, W. Shi, B. Carbunar, Y. Jiang, and N. Nguyen. 2012. Continuous mobile authentication using touchscreen gestures. In *Homeland Security (HST), 2012 IEEE Conference on Technologies for*. 451–456.
[18] M. Frank, R. Biedert, E. Ma, I. Martinovic, and D. Song. 2013. Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. *Information Forensics and Security, IEEE Transactions on* 8, 1 (2013), 136–148.
[19] GlobalWebIndex. 2014. 54% of tablet users share their device with others. (2014). https://www.globalwebindex.net/blog/tablet-sharing.
[20] B. Hoh and M. Gruteser. 2005. Protecting location privacy through path confusion. In *SecureComm*. 194–205.
[21] A. Jain and V. Kanhangad. 2015. Exploring orientation and accelerometer sensor data for personal authentication in smartphones using touchscreen gestures. *Pattern Recognition Letters* (2015).
[22] A. Karlson, B. Meyers, A. Jacobs, P. Johns, and S. Kane. 2009. Working overtime: Patterns of smartphone and PC usage in the day of an information worker. In *Pervasive Computing*. 398–405.
[23] L. Li, X. Zhao, and G. Xue. 2013. Unobservable Re-authentication for Smartphones.. In *NDSS*.
[24] W. Meng, D. Wong, S. Furnell, and J. Zhou. 2015. Surveying the Development of Biometric User Authentication on Mobile Phones. (2015).
[25] Y. Meng and D. Wong. 2014. Design of touch dynamics based user authentication with an adaptive mechanism on mobile phones. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. 1680–1687.
[26] Y. Meng, D. Wong, and R. Schlegel. 2013. Touch gestures based biometric authentication scheme for touchscreen mobile phones. In *Information Security and Cryptology*. 331–350.
[27] A. Mitchell, T. Rosenstiel, and L. Christian. 2012. Mobile Devices and News Consumption: Some Good Signs for Journalism. (2012).
[28] G. Montañez, R. White, and X. Huang. 2014. Cross-device search. In *KDD*. 1669–1678.
[29] YC Hacker News. 2017. Hacker News. (2017). https://news.ycombinator.com/.
[30] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. 2013. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *IEEE S&P*. 541–555.
[31] OFCOM. 2012. UK is now texting more than talking. (2012). http://media.ofcom.org.uk/news/2012/uk-is-now-texting-more-than-talking/.
[32] P. Pearce, A. Felt, G. Nunez, and D. Wagner. 2012. Addroid: Privilege separation for applications and advertisers in android. In *CCS*. 71–72.
[33] A. Pons and P. Polak. 2008. Understanding user perspectives on biometric technology. *Commun. ACM* 51, 9 (2008), 115–118.
[34] C. Romesburg. 2004. *Cluster analysis for researchers*.
[35] N. Sae-Bae, N. Memon, K. Isbister, and K. Ahmed. 2014. Multitouch gesture-based authentication. *Information Forensics and Security, IEEE Transactions on* 9, 4 (2014), 568–582.
[36] Google Play Store. 2015. HackerNews Reader. (2015). https://play.google.com/store/apps/details?id=com.xw.hackernews&hl=en.
[37] Robert L Thorndike. 1953. Who belongs in the family? *Psychometrika* 18, 4 (1953), 267–276.
[38] T. Vu, A. Ashok, A. Baid, M. Gruteser, R. Howard, J. Lindqvist, P. Spasojevic, and J. Walling. 2012. Demo: user identification and authentication with capacitive touch communication. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*. 483–484.
[39] Y. Wang, X. Huang, and R. White. 2013. Characterizing and supporting cross-device search tasks. In *WSDM*. 707–716.
[40] D. Watts and S. Strogatz. 1998. Collective dynamics of 'small-world' networks. *Nature* 393, 6684 (1998), 440–442.
[41] H. Xu, Y. Zhou, and M. Lyu. 2014. Towards continuous and passive authentication via touch biometrics: An experimental study on smartphones. In *SOUPS*, Vol. 14. 187–198.
[42] Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman. 2011. Identifying diverse usage behaviors of smartphone apps. In *IMC*. 329–344.
[43] Z. Yan, V. Subbaraju, D. Chakraborty, A. Misra, and K. Aberer. 2012. Energy-Efficient Continuous Activity Recognition on Mobile Phones: An Activity-Adaptive Approach. In *2012 16th International Symposium on Wearable Computers*. 17–24. DOI:http://dx.doi.org/10.1109/ISWC.2012.23
[44] J. Zhu, P. Wu, X. Wang, and J. Zhang. 2013. Sensec: Mobile security through passive sensing. In *ICNC*. 1128–1133.
[45] A. Zimek, E. Schubert, and H. Kriegel. 2012. A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining* 5, 5 (2012), 363–387.

Received May 2017