MobileMiner: Mining Your Frequent Patterns on Your Phone

Vijay Srinivasan
Samsung Research America
v.srinivasan@samsung.com

Saeed Moghaddam
Samsung Research America
s.moghaddam@samsung.com

Abhishek Mukherji
Samsung Research America
a.mukherji@samsung.com

Kiran K. Rachuri
Samsung Research America
k.rachuri@samsung.com

Chenren Xu*
WINLAB, Rutgers University

Emmanuel Munguia Tapia
Samsung Research America

ABSTRACT

Smartphones can collect considerable context data about the user, ranging from apps used to places visited. Frequent user patterns discovered from longitudinal, multi-modal context data could help personalize and improve overall user experience. Our long term goal is to develop novel middleware and algorithms to efficiently mine user behavior patterns entirely on the phone by utilizing idle processor cycles. Mining patterns on the mobile device provides better privacy guarantees to users, and reduces dependency on cloud connectivity. As an important step in this direction, we develop a novel general-purpose service called MobileMiner that runs on the phone and discovers frequent co-occurrence patterns indicating which context events frequently occur together. Using longitudinal context data collected from 106 users over 1-3 months, we show that MobileMiner efficiently generates patterns using limited phone resources. Further, we find interesting behavior patterns for individual users and across users, ranging from calling patterns to place visitation patterns. Finally, we show how our co-occurrence patterns can be used by developers to improve the phone UI for launching apps or calling contacts.

Author Keywords

Mobile Data Mining; Rule Mining; Context Prediction

ACM Classification Keywords

H.4.m Information Systems Applications: Miscellaneous

INTRODUCTION

Smartphones can collect and infer rich contextual data about users and how they use their phones. Using public APIs and apps on most smartphone platforms [3, 5, 30], we can easily log raw contextual data about the user such as her location, application usage, online activity, call and SMS behavior, charging behavior, and battery usage. Also, a large volume of research in the Ubicomp community has been devoted to using the raw location and physical sensor data to infer and log higher level user context such as biking or driving [23], or at a coffee shop [22]. In this work, we focus on the orthogonal, but equally important problem of inferring higher level *behavior patterns* from a longitudinal log of the raw and inferred context data collected by smartphones.

Figure 1. Example co-occurrence patterns and their uses.

User behavior patterns can be expressed in a number of different ways [2, 4, 34]. For example, sequence mining algorithms may uncover sequences of user contexts that occur frequently [34], while statistical correlation functions may express interesting relationships between numerical context data such as activity level and sleep quality [13]. Our long-term vision is to use longitudinal smartphone context to infer diverse frequent patterns that capture different aspects of the user's behavior, and explore the utility of each type of pattern in improving overall user experience. A key aspect of our vision is to leverage the computing potential of modern smartphones to perform the *pattern mining entirely on the device*.

Why Mine Co-occurrence Patterns?

In this work, as a first but important step towards our vision, we present a novel general-purpose service called MobileMiner that runs on the phone and discovers frequent co-occurrence patterns indicating which context events frequently occur together. Figure 1 shows illustrative co-occurrence patterns for an example user. For example, the first pattern could be expressed in an association rule [2] as $\{Morning, Breakfast, AtHome\} \rightarrow \{ReadNews\}$, and

* Chenren Xu worked on this paper as an intern at Samsung Research America.

UbiComp '14, September 13 - 17, 2014, Seattle, WA, USA

http://dx.doi.org/10.1145/2632048.2632052

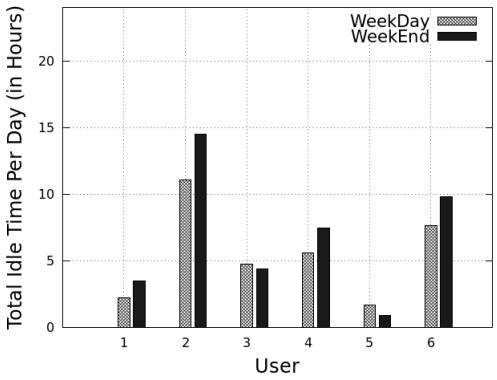


Figure 2. Average smartphone idle time per day for six sample users.

indicates that the user typically reads news apps on the phone whenever she has breakfast at home in the morning. Such a pattern could be used preload news content and news apps to memory ahead of time to reduce loading delays and improve user experience. Similarly, as seen in Figure 1, we can provide convenient UI shortcuts to typical user tasks performed at work in the afternoon, such as calling Alex, or opening a project folder. Finally, we can even try to expose and alter frequent patterns that are not beneficial to the user; for example, Figure 1 shows a smart reminder to charge the phone before the user typically goes to sleep with a half-empty battery. A key benefit of using association rules is that they can be easily read and understood by both end users and developers, and even be used in simple *if-this-then-that* style [14] mobile recipes, compared to potentially more accurate but less readable classification models.

Why Mine Patterns on the Mobile Device?

Modern smartphones have powerful quad-core processors [10] and are also typically unused for a majority of time such as at night when the user is sleeping and the phone is charging. Compute-intensive pattern mining algorithms may be run on the phone periodically during this idle time with little or no impact on the end user. Figure 2 shows the average *idle time* per day on weekdays and weekends for 6 users over 2 months of smartphone usage. We define the phone to be *idle* whenever it is charging, there are no foreground applications, and the battery level is at least 80%. Figure 2 shows that users have between 1-10 hours of idle time each day, which may be aggregated across multiple days for mining algorithms that do not need to refresh their patterns every day. In fact, we show later in the paper that multiple instances of mining algorithms may be run simultaneously on a quad-core smartphone with little or no impact to user experience, thus increasing the mining capacity during the idle time. Another key reason to run mining algorithms on the phone is to provide better privacy guarantees to the user by ensuring that personal context data is not transmitted to the cloud for mining. Our approach is complementary to parallel efforts to improve the privacy guarantees of sharing personal context data with the cloud [20]. Finally, mining patterns on the device may also provide benefits to users with lower-end phones in developing or remote regions, where cloud connectivity and data plans may be limited or absent.

In this paper, we present four main contributions.

Our first key contribution is the design and implementation of the MobileMiner pattern mining service. MobileMiner runs entirely on the phone and mines the user's co-occurrence patterns using limited phone resources. MobileMiner is extensible in allowing new context streams to be added, and is easily configurable to mine patterns at multiple resolutions, such as discover *overall* frequent patterns of a user or more detailed patterns relating to specific context types such as *app usage* or calls. As part of MobileMiner, we develop an optimized weighted mining algorithm that efficiently mines patterns by leveraging the repetitive nature of mobile context data. MobileMiner also provides a prediction engine that allows developers to use the co-occurrence patterns to retrieve predictions about user context in the near future.

Our second major contribution is the experimental validation of the feasibility of running the MobileMiner implementation on a phone using context logs that we collected from 106 users over 1-3 months of data. MobileMiner mines overall user patterns over 1-3 months of data in 16.5 minutes, consumes 0.01-3% of the phone's battery charge per day, and is 15 times faster than a highly optimized version of the widely used Apriori algorithm for association rule mining. Further, our configurable approach mines detailed behavior patterns about specific context types such as app usage in 21 seconds.

Thirdly, we analyze patterns from individual users to understand their utility in multiple use cases, and also analyze which patterns occur frequently across all our 106 users. Commonly occurring patterns could be used to uncover and express common sense knowledge about user context, and also to improve user experience during the cold start period when personal behavior patterns are being learned. Further, we cluster users into groups based on their co-occurrence patterns, and visualize patterns from individual groups to explore the utility of learning common patterns in groups of users.

Finally, we implement two UI improvements using predictions from MobileMiner. We predict the next outgoing call or app launch, and provide users with convenient predictive shortcut icons for the next contacts to be called or the next apps to be launched. We outperform a majority predictor that simply uses frequency counts. Further, we allow end users to configure the number of app or call recommendations, and the recall-precision tradeoff for predictions based on their individual preferences. We conclude with a survey over 42 users that probes the utility of our app prediction service and the desired operating points for individual users.

SYSTEM DESIGN

Our MobileMiner service was implemented on the Tizen Mobile platform for smartphones [30]. Tizen is an open and flexible operating system that is under open source governance, open to all members who wish to participate, and resides within the Linux Foundation [19]. The Tizen operating system comes in multiple profiles to serve different industry

requirements: Tizen IVI (in-vehicle infotainment), Tizen Mobile used in smartphones, Tizen TV, and Tizen Wearable used in Samsung Gear [11].

Figure 3 shows the overview of our MobileMiner system architecture. We formulate the co-occurrence pattern mining problem as the well-known association rule mining problem [2] from market basket data. From the raw context data on the phone, we extract a sequence of timestamped baskets using the **base basket extractor**, where each basket indicates which contexts occur together at a given timestamp; for example, the basket $\{AtHome, ReadNews, Morning\}$ could be followed by $\{AtHome, Morning, ChargePhone, PlayPandora\}$. These baskets are input to a **base rule miner** that outputs frequent co-occurrence patterns expressed as association rules. Each association rule is of the form $A \rightarrow B$, where A and B are co-occurring sets of context events, and the association rule indicates that the context events in B are likely to occur whenever we observe the context events in A. For the association rule $A \rightarrow B$, we define A as the antecedent and B as the consequent. For rule $A \rightarrow B$, we also define two parameters:

- **Support:** Support represents the fraction of times the context set A, B occurs in input baskets; in other words, the joint probability $P(A, B)$.

- **Confidence:** Confidence represents the proportion of times B co-occurs in the same basket whenever A co-occurs in a basket; in other words, the conditional probability $P(B|A)$

We apply a support and confidence threshold for each instance of the rule miner, indicating the minimum support and confidence of each rule discovered. Reducing the support results in more patterns but also increases the running time of the rule mining algorithm. Rule mining algorithms, especially optimized versions of the Apriori algorithm [8], have been applied widely to supermarket *basket data* in order to discover association rules between different items purchased in a transaction. However, when Apriori is applied to mobile context data, we observed a high running time spanning several hours. To address this challenge, we developed a weighted mining algorithm called **WeMiT** (Weighted Mining of Temporal Patterns) that leverages the repetitive nature of mobile context to significantly reduce the running time of Apriori.

The base pattern miner shown in Figure 3 mines patterns over a large number of base baskets spanning several months. Therefore, we use a high support threshold to achieve a feasible running time and discover user patterns over long duration activities such as phone charging or place visitation. However, if we are interested in patterns about user contexts that are instantaneous or short-lived, such as launching apps or making calls, we need to use a low support threshold over a large number of base baskets, resulting in an unfeasibly high running time. To address this challenge, using APIs exposed by MobileMiner, we use an **app usage filter** to retrieve relevant baskets of interest, such as only baskets containing app launching context. We then mine rules specific to app usage as shown in the **app rule miner** in Figure 3. Similar filters

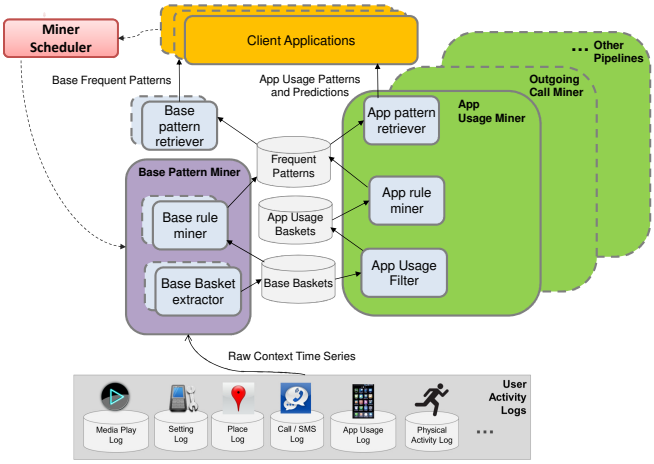


Figure 3. MobileMiner system architecture.

and rule miners may be instantiated for other contexts such as outgoing calls or online activity.

External client applications use our **pattern retriever** component to not only retrieve overall and detailed patterns, but also retrieve predictions about contexts in the near future based on the MobileMiner **prediction engine**. In the rest of this section, we elaborate in more detail about the basket extraction and filtering component, the rule mining algorithm, and the prediction engine used to predict future contexts based on co-occurrence patterns. In our current implementation, client applications use our miner scheduler to periodically schedule the mining algorithm whenever the phone is idle.

Basket Extraction and Filtering

We first extract co-occurrence baskets from the diverse context data streams input to the co-occurrence engine. As the frequent pattern mining algorithm works on categorical data, we discretize continuous context data to a set of small discrete values; for example, location is discretized into major location categories such as home, work, and outside, and battery levels to 10 equal-sized bins from 0-100.

Base Basket Extraction: The basket extraction process is illustrated in Figure 4 using a simple example. Each context data stream is composed of a sequence of interval events with a categorical context item. The basket extractor module creates a sequence of timestamped context baskets based on temporal overlap between the interval events. Each context basket consists of a set of context names that occur together at a given timestamp. In Figure 4, we show some sample time instants indicated by the vertical dashed lines and the corresponding context baskets created below for each time instant. We see that $\{Morning, AtHome, ListenToJazz, ReadComics\}$ is recorded as a context basket denoting that when the user was at home on a Monday morning, she listened to jazz music on her phone while reading comics. During the extraction process, we add additional derived items such as day of week, time of day at hour granularity, and time of day at a higher granularity such as morning or evening. The goal is to generate generalized rules where possible, as a set of events

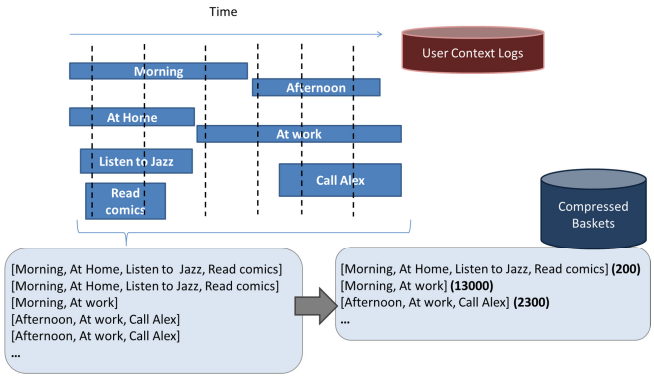


Figure 4. Compressed Basket Extraction.

may not necessarily co-occur at the same exact hour of day, but may frequently co-occur in the morning or on weekdays.

The default sampling interval used to create the itemsets (distance between dashed black lines in Figure 4) is 30 seconds. Due to the repetitive nature of context data, we observe that basket extraction leads to several duplicate baskets. For example, if the sampling interval is 30 seconds, and the set of items $\{Morning, AtHome, ListenToJazz, ReadComics\}$ happens for 100 minutes over 3 months, then there will be 200 such context baskets. Instead of repeating the baskets, we compress each basket into a *compressed or weighted basket* and associate a weight with each compressed basket, indicating the number of times it repeats. Examples of compressed baskets are shown in Figure 4. These weighted baskets are used by our **WeMiT** algorithm to achieve a faster execution time compared to the widely used Apriori algorithm. Also, our basket extractor is configurable to compute baskets over larger sampling intervals of 30 minutes to 3 hours, enabling us to extract baskets with co-occurring context events separated by longer time durations; for example, an *SMS to contact1* occurs within 10 minutes of a *phone call to contact2*. Detailed analysis of the utility of patterns retrieved using longer sampling intervals is deferred for future work.

Basket Filtering: Using the basket filtering component, we allow developers to filter baskets of interest to their application domain. As mentioned above, the filtering step is used to improve the efficiency of the rule mining algorithm when discovering detailed rules about specific context types such as app launches. Developers specify the set of baskets they are interested in using a boolean expression over context items. For example, developers may retrieve only baskets relating to app usage as shown by the app usage filter in Figure 3, or specify a complex boolean expression such as all baskets with no foreground application, battery level greater than 80%, and phone charging. We also provide utility functions in the basket filtering component, including functions for adding new *last context* features to baskets such as the last app used or the last contact called. We found these features to be especially useful in improving the accuracy of call or app predictions using MobileMiner patterns. In general, mining rules over a filtered subset of baskets is faster compared to mining rules from the complete base baskets. Also, rules mined from the filtered baskets are free of the noise from the rest of the data.

Weighted Rule Mining

The weighted rule mining algorithm accepts as input weighted context baskets and outputs association rules of the form $A \rightarrow B$. The rule miner first generates **frequent itemsets** of all sizes, indicating which sets of context events frequently occur together. An itemset is defined as *frequent* if it occurs at least as many times as the *support threshold* defined earlier. An example frequent itemset is: $\{AtHome, UsingWiFi, Between10-11pm, ChargingPhone\}$. From the frequent itemset, all possible combinations of association rules of the form $A \rightarrow B$ are generated, as long as the rule's confidence exceeds the *confidence threshold* defined earlier. An example association rule generated from the above frequent itemset is: $\{AtHome, UsingWiFi, 10-11pm\} \rightarrow \{ChargingPhone\}$; this rule could help predict if the user is likely to charge the phone given the current context. A key challenge in association rule mining is to efficiently discover the frequent itemsets; from the frequent itemsets, we found it computationally inexpensive to evaluate all possible association rules with a limited number of consequents.

Formally, let $I = \{i_1, i_2, , i_n\}$ be a set of context items, such as *At Work* or *Monday*. Let $B = \{b_1, b_2, , b_n\}$ denote the context baskets generated using the basket extractor, such as $\{AtHome, Morning, Monday\}$. Given B, the goal of frequent itemset mining is to discover all subsets of context items I co-occurring more than a threshold number of times denoted by the support threshold; the resulting subsets of I are called frequent itemsets. Using a lower support value increases the running time of the mining algorithm and generates a higher number of frequent itemsets.

Apriori Algorithm: We first evaluate an optimized version of the Apriori algorithm [2] for frequent itemset mining. The Apriori algorithm is a "bottom up" mining approach that first generates frequent itemsets F_n of size n (initialized to 1) and then repeatedly generates itemsets F_{n+1} of size $n+1$ by using the downward closure property of support which states that all subsets of a frequent itemset are also frequent. To evaluate if a candidate itemset is frequent, we make one pass through the entire set of baskets B to check if its frequency exceeds the support threshold. A naive implementation of the Apriori algorithm evaluates $|I_{freq}|.|F_n|$ candidate itemsets at each $(n+1)$th iteration of the algorithm, where I_{freq} denotes the set of singleton frequent itemsets. In our optimized version, for each candidate itemset X in the $(n+1)$th iteration of the algorithm, we check if each subset of X of size n is already a frequent itemset generated in the previous iteration. If there is at least one subset that is not frequent, candidate itemset X is pruned; thus, we avoid the need to make a single pass through the entire basket set B for the pruned itemsets, reducing running time significantly. However, the optimized Apriori algorithm still requires several hours when mining longitudinal baskets over several months, thus motivating the need for more efficient mining algorithms suitable for a smartphone.

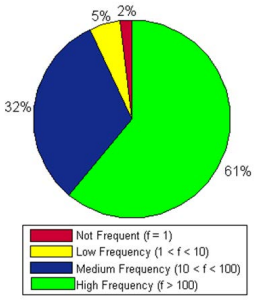


Figure 5. Distribution of basket frequencies for a sample user.

WeMiT Algorithm: In typical itemset mining problems suc as supermarket basket mining, the input co-occurrence iten sets are slightly different; for example, the chances of buyin exactly the same items at the same time and location are lov However, as Figure 5 shows for a sample user, many mobile context baskets are repeated exactly due to the slow changing nature of user context and activities; only 2% of itemsets are unique and 61% of itemsets occur more than 100 times.

On average, we observed a 92.5% decrease in the number of the compressed weighted baskets compared to the number of uncompressed baskets extracted in the base basket extractor; we leverage this by using only the weighted baskets as input for WeMiT. Since the weighted baskets are smaller, the running time of frequent itemset mining, which requires several passes through all the baskets to evaluate candidate itemsets, is greatly reduced. In particular we use the weighted baskets $B = \{b_1^{w_1}, b_2^{w_2}, , b_n^{w_n}\}$, where w_k is the weight of basket b_k, and modify the definition of support of an itemset X in the weighted baskets B to be: $\sum_{i=1}^{n} contain(b_i^{w_i}, X).w_i$, where $contain(p, q) = \{1 \text{ if } q \subseteq p \mid 0 \text{ otherwise }\}$. The downward closure property of support used in Apriori is true for the new definition of weighted support as well. We implement the WeMiT algorithm on Tizen phones and show that WeMiT reduces the running time of our optimized Apriori algorithm by 15 times on average.

Context Prediction

The co-occurrence patterns generated by the MobileMiner algorithm may be used to predict future context events, such as the next app used or the next contact called. While there is a large body of work on using association rules for classification and prediction [33, 17], we implement a practical prediction algorithm that is designed to return prediction results within a second on resource-constrained smartphones.

Figure 6 shows how context prediction is performed using co-occurrence patterns. The prediction engine is input the current context (say $\{Morning, AtWork\}$) and also the target context to be predicted such as the next app used or next contact called. Given this input, we search the association rules generated by MobileMiner to find rules where the antecedents are a subset of the current context and the consequent matches the target context category; an example matching rule is: $\{Morning\} \rightarrow \{UseGmail\}$ with confidence 0.9. We return a list of predictions based on the multiple contexts found in the consequents of matching associ-

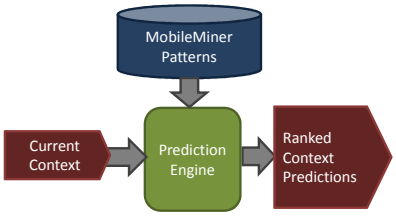


Figure 6. Context prediction using co-occurrence patterns.

ation rules. For example, if we find another matching rule $\{AtWork, Morning\} \rightarrow \{UseOutlook\}$ with confidence 0.9, we return both the Gmail and Outlook app predictions. We rank the predicted contexts based on the decreasing order of confidence values for each unique prediction, where the confidence value for a prediction is computed as the maximum confidence value among all matching rules found for that prediction. For example, the confidence value for the Gmail prediction would be 0.9 even if there is another rule $\{AtWork\} \rightarrow \{UseGmail\}$ with confidence 0.8. When the confidence values for two different predictions match, we favor rules with a higher intersection of context items between the rule antecedent and the current context; in our simple example above, we favor the Outlook app over the Gmail app prediction, since the matching association rule for Outlook has two antecedents in common with the current context, as opposed to one for Gmail.

EVALUATION

Context Data Collection

To evaluate our MobileMiner service, we collect mobile context data from 106 participants recruited. We develop an Android application called EasyTrack using the Funf sensing library [3] to collect anonymized and encrypted context data. We used commercial crowd-sourcing platforms to recruit users for our data collection [31]; we had healthy mix of gender and occupation over our 106 participants. On average, for each user, we observed 440 unique context events, including apps used, contacts called, or locations visited. For evaluation purposes, we included the following timestamped context events: (1) call events including call type (incoming, outgoing, missed, voice mail), duration, and number, (2) SMS events including SMS type and number, (3) inferred place identifiers of home, work and outside, (4) location cluster label for the current location obtained using a simple location clustering algorithm, (5) phone charging status, (6) battery levels, (7) foreground app usage events, (8) WiFi or cell connectivity, (9) cell ID of the current location, (10) binary movement status of the user.

While our entire data collection effort lasted more than 6 months, we observed that some users were more active in the data collection, while other users participated only for a few days before disabling or uninstalling the application. In Figure 7, we show the distribution of the number of full days of data without any data loss that we effectively collected from our 106 users. At least 40 users collected more than 40 days of full data, while around 25 users only collected 21-40 days of data.

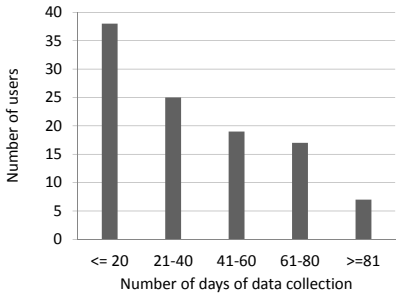


Figure 7. Distribution of number of full days of data collected from our 106 users.

Performance Metric	Base Basket Extraction	Base Rule Mining	App Usage Filtering	App Usage Rule Mining
Execution time	1.7 seconds	16.5 minutes	1.4 seconds	21.2 seconds
Memory	9.9 MB	44.2 MB	11.6 MB	1.0 MB
CPU Utilization	22.9 %	24.3 %	20.8 %	21.9 %
Number of baskets or rules	114275 baskets 8559 compressed	46675 rules	752 baskets 327 compressed	1062 rules
Energy per day as % of full battery	<0.01 %	0.45 %	<0.01 %	0.01 %

Figure 8. MobileMiner Performance.

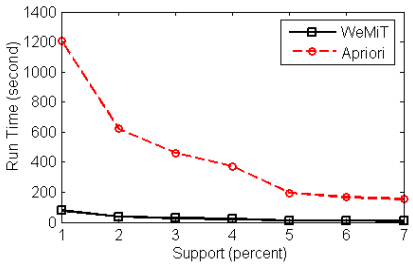


Figure 9. Comparison of WeMit performance with Apriori.

System Performance

We measure the performance of our MobileMiner system on a Tizen phone with the equivalent hardware configuration of the Samsung galaxy S3. Our goal was to answer the three main questions explained below in detail.

Is it feasible to run MobileMiner components on the phone?

To answer this question, we measure the running time, CPU utilization, and memory consumption of individual MobileMiner components for a representative subset of 28 users from our full set of 106 users with 2-3 months of context data. Figure 8 shows the results for four important MobileMiner functions: (1) the base basket extraction, (2) the base rule mining used to mine overall patterns of users, (3) the basket filtering for an example application of app usage pattern mining, and (4) the rule mining of app usage patterns. We used a relative support threshold of 1% of all base baskets for base rule mining, and an absolute support threshold of 20 occurrences for the app usage rule mining.

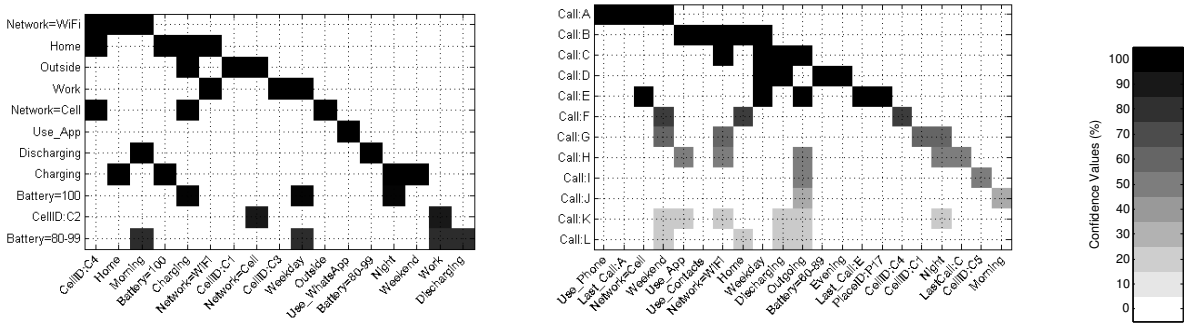
We observe that base basket extraction takes less than 2 seconds to process one day of context data; this is because the extraction process is incremental and needs to access only the past day's data. We observe on average that compression reduces the number of baskets by 92.5%, which improves the running time of our WeMiT algorithm compared to the widely used Apriori algorithm. The base rule mining is the most time consuming of our components, and requires 16.5 minutes on the phone to mine base patterns over 2-3 months of data. However, given our observations about the idle time on smartphones from Figure 2, users can easily accommodate 17 minutes of phone resources for mining once every 2-3 days. The memory consumption of rule mining is moderately high, but still acceptable since it is run when the phone is idle and there are no foreground applications. We observe more than 46000 rules generated by the rule miner. Our pattern retriever APIs allow developers and end users to retrieve much smaller sets of patterns based on their domain of interest, confidence and support threshold. Also, our prediction engine automatically combines information from all generated rules to provide an accurate prediction of future context.

From Figure 8, we also observe that the filtering of app usage baskets takes less than 2 seconds per day, and the total time required to mine fine-grained user patterns about app usage is only 21.2 seconds. Thus, rule mining on filtered subsets of baskets is an efficient approach for developers to mine patterns based on their domain of interest. Since the app usage baskets are instantaneous events, we observe only a 55% compression in the number of baskets; however, given the small number of uncompressed app usage baskets, the running time of app rule mining remains low.

Also, we observe from Figure 8 that the CPU utilization of MobileMiner components is less than 25%. Based on empirical experiments, we were able to simultaneously execute 3 instances of MobileMiner on our smartphone device with no perceptible reduction in frames per second for a popular image processing app. This shows that instances of mining algorithms operating on different context domains such app usage or outgoing calls could be run simultaneously during the phone's idle time to increase the mining capacity.

Finally, in Figure 8, we report the energy consumed per day by MobileMiner as a percentage of the Galaxy S3's full battery charge (7.98Wh), assuming we perform the mining once per week. We use the Monsoon power meter [24] to measure the power consumption of each component across five sample users. If we run the mining once per week, discovering base and app usage rules consumes 0.45% and 0.01% respectively. Even if we run the mining once per day, discovering base and app usage rules consumes only 3.09% and 0.05% respectively. For the majority of users, the *effective* power consumption of mining could be zero or negligible, since the mining is performed during the surplus time when the phone is plugged in and fully charged. Since we are interested in long-term patterns, we envisage running the co-occurrence engine only infrequently, such as once per day or week. In the future, we propose to estimate the appropriate frequency of mining based on the rate of change of user patterns and the application needs.

How does WeMiT compare to Apriori?



(a) Overall patterns. (b) Outgoing call patterns. (c) Key.

Figure 10. Co-occurrence patterns for sample user 38. We show the confidence of each rule in the matrix visualization.

We evaluate the impact of using compressed baskets and our weighted mining algorithm on the performance of base rule mining. Since the running time of Apriori is of the order of several hours for some users, we restrict ourselves to a subset of 13 users to compare Apriori with WeMiT. Figure 9 shows the results of our running time comparison for support values ranging from 1% to 7%. For lower support values, which is of higher interest to uncover patterns about important short duration activities such as walking or exercising, we observe that WeMiT is 15 times faster than the Apriori algorithm. For this subset of users, Apriori requires a running time of 20 minutes on the phone, while WeMiT discovers the same patterns in 78.5 seconds. Thus, we observe that compressing and using the weighted baskets for mining mobile context data significantly reduces the running time of pattern mining.

Patterns Generated

We visualize sample co-occurrence patterns generated from our context logs over 106 users with the goal of answering the two main questions below.

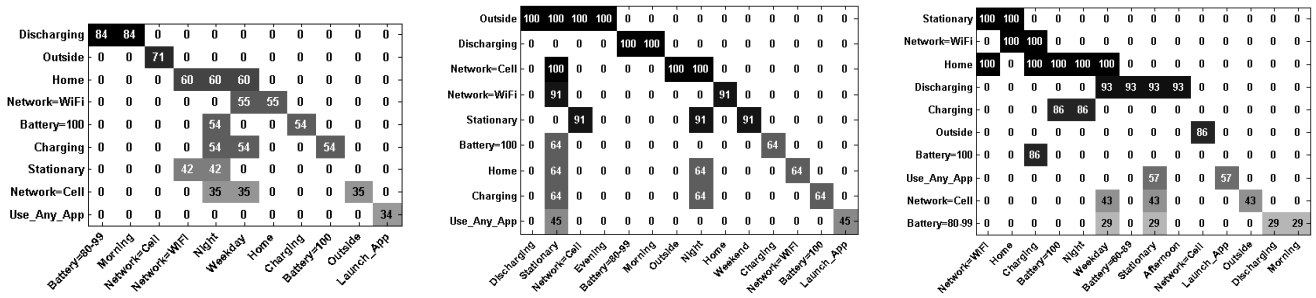
What are some sample patterns and how do we use them?
We visualize a small subset of association rules generated for a sample user to analyze the patterns generated by MobileMiner. Figure 10 shows sample patterns generated for user 38 from our dataset of 106 users. Figure 10a shows base patterns mined by the base pattern miner, and Figure 10b shows detailed patterns about outgoing calls made by the user. Each row of the matrix visualized in Figure 10 represents an association rule. The row name corresponds to the consequent of the rule, while the column names correspond to the antecedents of the rule. For each row, we color a cell only if the association rule contains the antecedent specified in the column and the consequent specified in the row. Cells are color coded based on the confidence value expressed in percentage as shown in Figure 10c.

Figure 10a shows base patterns for user 38 for long duration contexts such as Wi-Fi connectivity or home occupancy. Due to space constraints, we display only one sample pattern per consequent for this visualization and limit ourselves to 10 rules. We use a support threshold of 1% to generate these patterns. While there may be several rules for WiFi connectivity for user 38, the first row shows a sample rule which indicates WiFi connectivity whenever she is at home in the morning in cell ID C4. However, from row 5, we see that when she is in the same cell ID C4 but outside her home and charging her phone, she is connected to a cell network and not WiFi. Similarly, row 2 shows that she is at home whenever she is in cell ID C4 in the morning, and her phone is plugged in and fully charged. Such patterns can be used in several applications including: (1) preloading data intensive content when she expects to have only cell connectivity in the near future, (2) sensing expensive contexts such as place identifiers using cheaper attributes such as charging status and battery levels, and (3)providing reminders to set a low power mode or charge the phone based on typical charging and discharging patterns.

Figure 10b shows the outgoing call patterns for user 38, which could be used to provide shortcut icons for the next contact likely to be called. From Figure 10b, we see that user 38 calls user A and user E again after calling them last. Also, user 38 calls user A on weekends when connected to a cell network, while she calls user B on weekdays when connected to Wi-Fi at home. In the next section, we will show how we use co-occurrence patterns to predict the next outgoing call and improve upon the common approach of showing frequently called contacts in popular phone platforms [5, 15].

What are some common patterns across multiple users?
We analyze commonly occurring co-occurrence patterns across multiple users, and how we can potentially use them. Figure 11 shows common co-occurrence patterns across multiple users using the same matrix visualization in Figure 10 as above; however, the main difference is that in Figure 11, in each cell, rather than the confidence, we show the percentage of users the pattern occurs in, either among all users (Figure 11a), or among smaller groups of users with very similar co-occurrence patterns (Figures 11b and 11c). We only show patterns that have at least 80% confidence 1% support.

Figure 11a shows the most common patterns across all our 106 users. The most frequent pattern occurring in 84% of 106 users is a common sense pattern that indicates the phone is discharging if the phone battery level is high in the morning, as the user wakes up, unplugs the phone and begins to use it. Another common pattern is that the user is typically outside of home and work when connected to a cell network. Such common patterns could be used to bootstrap an initial



(a) All users. (b) Sample group 1. (c) Sample group 2.

Figure 11. Common rules across multiple users. For each rule, we show the percentage of users the rule occurs in, either among all users (Figure 11a), or among smaller groups of users with very similar co-occurrence patterns (Figures 11b and 11c).

set of patterns while MobileMiner slowly learns personal co-occurrence patterns over time. Common patterns could be a useful way to express and use common sense knowledge about user context from a particular community of users, such as office goers or students.

Additionally, we cluster users using the k-means clustering algorithm [12] based on the number of common patterns between any pair of users. Figures 11b and 11c show the common co-occurrence patterns for two groups of users from this clustering; we show the percentages of users each pattern occurs in among users in each group. We see that all group 1 users are stationary when connected to a cell network on weekend nights while all group 2 users are stationary when at home and connected to Wi-Fi. Group 1 users are usually outside when connected to a cell network in the evening, while group 2 users are usually at home when connected to Wi-Fi on weekday nights. In the future, we will explore how these group patterns could be used for group activity scheduling, or extrapolating patterns from a group to an individual to enable recommendation services.

EXAMPLE USE CASES: APP AND CALL PREDICTION

We use the co-occurrence patterns and the prediction engine of our MobileMiner service to implement two example use cases. We predict the next contact called or the next app launched by the user and provide convenient UI shortcuts for likely user actions. The main benefit of such a service is to lessen the burden of searching for the contact or app on the phone. Figure 12 shows two screenshots of our app recommendation service implemented on Tizen. We display the recommended apps as shortcuts in the quick panel. We allow users to set the number of shortcut icons they would like to receive. As seen in Figure 12, a useful feature of our system is that we actually show users the pattern that was used to recommend each app, since co-occurrence patterns are human-readable; such a feature could be very helpful for users in understanding and debugging our app recommendation service. To evaluate the accuracy of app and call predictions, we use two standard evaluation metrics from the classification and prediction literature:

- **Recall** is the proportion of app launches or outgoing calls for which we show recommendations to the user
- **Precision** is the proportion of times the user uses one of our shortcut icons to complete his task

Typically, higher recall results in a lower precision and vice versa. In our system, we use a threshold on the confidence of the highest ranked prediction to decide whether or not to show recommendations to the user. Using a lower confidence threshold results in higher recall, and using a higher confidence threshold results in a lower recall and higher precision. Further, we allow users to configure the recall-precision tradeoff based on their individual preferences. Given the above experimental setup, we answer two main questions with our evaluation.

What is the Recall-precision Tradeoff of Our Predictions?

To evaluate the recall-precision tradeoff our services, we replay our MobileMiner algorithm on the longitudinal trace of app launch and outgoing call data with increasing confidence thresholds. Figures 13a shows the recall-precision tradeoff for app recommendations for 1,3,5, and 7 shortcut icons shown to the end user. We also compare our prediction approach against a majority predictor that predicts the most frequently launched apps, as used in commercial phone platforms [5, 15]. Figure 13a shows the app prediction results over all 106 users, while 13c shows the call prediction results over a smaller subset of 25 users with 1,3, and 5 shortcut icons for contacts; we limited our evaluation for call prediction because we eliminated a large number of users who had very limited call logs.

Figures 13a and 13c show that prediction based on co-occurrence pattern outperforms the majority classifier based on the metric of area under the recall-precision curve. When making predictions up to 50% of the time, in general, we achieve a 89-184% improvement in precision over the majority predictor in correctly predicting the next app launched or next contact launched. As will be discussed in Table 1 for app recommendations, users can select the operating point of interest to them on the recall-precision curve; some users prefer 90% precision with 3 app recommendations and a recall of 35%, while other users prefer a precision of 80% with 5 app recommendations and a recall of 68%.

Comparing figures 13a and 13c, we find that the prediction accuracy of both the majority and MobileMiner predictors

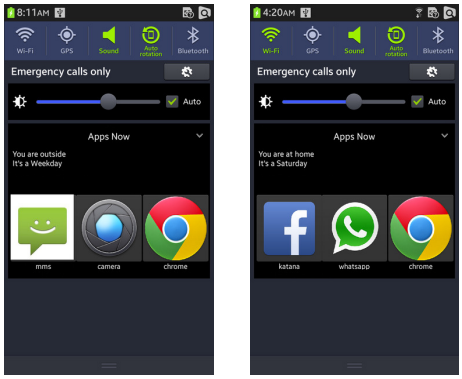


Figure 12. App recommendation service with shortcut icons. For each app, we show the reason why it was displayed based on the matching co-occurrence pattern.

is lower for call recommendations compared to app recommendations. The reason might be due to the smaller amount of call log data from which to learn co-occurrence patterns, while another reason could be that users have less predictable calling patterns compared to app usage patterns. Further exploration of this reason, and potential new basket features to improve call prediction accuracy is deferred for future work.

How do we choose the support value for mining patterns? An important challenge in rule mining approaches is to choose the appropriate support threshold to discover patterns. In Figure 13b, we show the effect of the absolute support threshold on the accuracy of MobileMiner app predictions for 3 app recommendations. On average, we see a 4-5% improvement in precision as the support decreases from 20 to 5. Thus, mining patterns that occur even only 5 times could be potentially useful in improving prediction accuracy. The average running time for MobileMiner for support values of 20, 15, 10, and 5 are 12.4, 37.1, 174.8, and 2218.2 seconds respectively. Developers can use both the recall-precision curves and the execution time of the miner to determine the appropriate support threshold for their use case. For example, a developer may choose a support value of 10 to achieve reasonable prediction accuracy without incurring considerable phone resources.

What do Users Think of Our App Prediction Service?

We conducted a user survey with 42 participants using an online survey tool [29] to collect feedback from real, external users not involved in the research effort about the usefulness of our app recommendation service and the performance expectations of users. We recruited a subset of 42 users from the 106 users used in our performance evaluation to answer this survey. One limitation of our survey is that the respondents did not actually use our app recommendation service, since it was implemented on Tizen phones that were not commercially available at the time the survey was conducted. In the future, we plan to deploy our app recommendation service in-situ over a large number of users to collect their real-time feedback in using our service, by porting our Tizen MobileMiner implementation to Android. Respondents answered the survey based on a clear explanation of our app recommendation service with screenshots of the app showing multiple recommendations. In the survey, we used the following

Table 1. Responses to the question: Based on the data collected, which of the following configurations would you prefer with regards to precision, number of recommendations, and recall?

Precision	No. of recommendations	Recall	Responses
90%	3	35%	30.95%
80%	3	51%	16.67%
80%	5	68%	23.81%
80%	7	80%	11.90%
75%	3	66%	4.76%
75%	5	87%	11.9%
75%	7	100%	19.05%

simplified explanations of recall and precision to ask about user preferences on the recall-precision tradeoff: (a) Recall was defined as: **recommendation frequency** - percentage of times that the service recommends apps to you, and (b) precision was defined as: **accuracy** - Percentage of times that you select one of the service's recommended apps.

Below we present the questions and results of our user survey. Overall, we found that users would respond positively to our app recommendation service. Further, we found that each user had different preferences for the recall-precision tradeoff and the number of app recommendations, which further supports our configurable approach of allowing users to choose the confidence level of each prediction.

How often will users use our app recommendation service? 57% of users said that they will use the service regularly. 42% felt that they will use the service sometimes, and all the respondents said that the service is useful.

Where should the shortcut icons be placed? 40% of users prefer to see the icons on their phones' lock screen, 26% on their phones' quick panel, and 33% prefer them in the main tool bar at the bottom of the home screen.

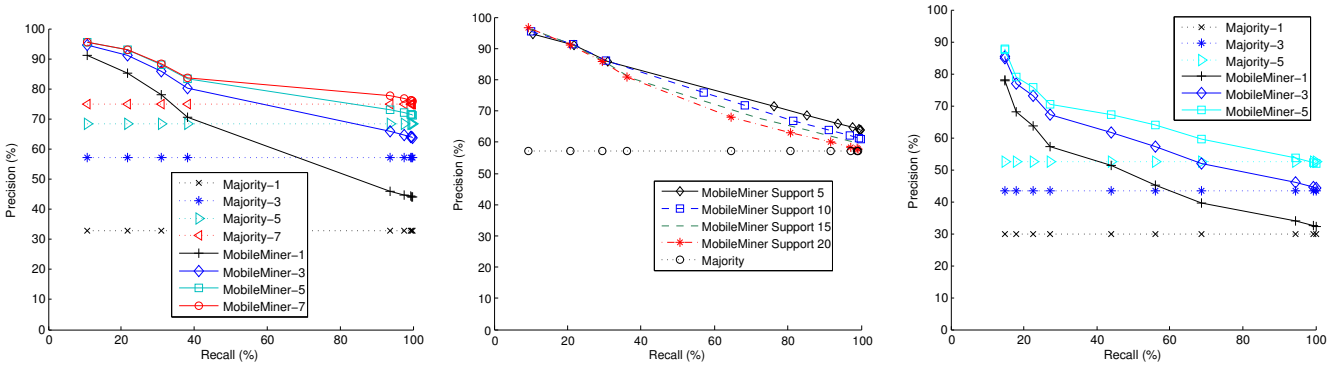
How many shortcut icons should be displayed? 71% of the survey respondents prefer to receive 4 to 6 icons, whereas 26% of the respondents would like to receive 1 to 3 icons. Very few of them prefer more than 6 predictive icons.

Will users prefer a recall less than 100% for improved precision? 54% of users prefer higher precision with recall less than 100%. 9% of users would prefer to always receive the recommendations and 35% said that either case (i.e., always showing recommendations or only when the confidence is high) would be acceptable.

What is recall-precision tradeoff preferred by users? Based on our recall-precision tradeoff plots in Figure 13a, we asked users about their preferred operating points. We observed a considerable variation among the responses regarding user preferences for precision, number of recommendations, and recall. The results are summarized in Table 1. Thus, our approach of allowing users to configure this tradeoff based on their individual preferences could be valuable.

RELATED WORK

Association rule and frequent itemset mining has been recognized as an increasingly critical technology for data-intensive decision-making [1, 2, 12]. Most works have focused on im-



(a) App prediction.

(b) Effect of support on app prediction with 3 recommendations.

(c) Call prediction.

Figure 13. Recall-precision tradeoff for app and call prediction using co-occurrence patterns. Prediction using co-occurrence patterns significantly outperforms a majority predictor (Figures 13a and 13c). Decreasing support shows a marginal improvement in app prediction accuracy (Figure 13b).

proving the efficiency of mining algorithms [1, 2] for large scale data in the cloud or desktop environments. Yet, the usability of such algorithms on mobile devices remains questionable due to the high computational complexity and the need to transmit personal data to the cloud. In this work, we focus on this relatively unexplored problem of on-device mining of co-occurrence patterns over users mobile context data.

Context-aware computation on mobile devices has gained a lot of interest recently. Particularly, energy-efficient techniques for inferring a user's activities [16], locations [6, 18, 21], and proximity to others [7] have been proposed. However, only a few existing works have explored using longitudinal context data for mining typical user behaviors; an example is the ACE system (Acquisitional Context Engine) [25] that mines co-occurrence patterns among context events, and exploits the patterns to speculatively sense the users context in an energy-efficient manner, which could be another application of MobileMiner. However, unlike our work, ACE does not develop an optimized version of the rule mining algorithm that is run on the phone; instead, ACE pushes all context data to a remote server where the pattern mining takes several hours to complete, resulting in privacy, data cost, and latency concerns.

Some approaches use specialized predictive classifiers and other targeted approaches [27, 35, 26, 32] for phone optimizations such as app launching or app preloading. Compared to such an approach, our MobileMiner approach has several advantages. First, our approach is more generalizable: using the same set of patterns generated, we can predict the next app launched, the next contact called, the next web page visited, or even the next action within an app. Further, our approach provides more configurability by allowing users to trade off prediction accuracy with prediction selectivity by varying the confidence threshold. Also, since our patterns are exhaustive and include all possible subsets of antecedents, our system can gracefully make predictions with lower accuracy even with missing context events. Finally, our co-occurrence patterns are more readable and directly usable by end users in mobile recipes [14], compared to potentially more accurate but less readable classifiers.

A preliminary version [28] of our work proposing the basic idea of on-device mining of mobile user patterns has been presented as a poster, without presenting details about the detailed system design, evaluation, the patterns generated, and the use cases of app and call prediction.

CONCLUSIONS

In this paper, we presented the novel MobileMiner system for mining frequent co-occurrence patterns on the phone indicating which context events frequently occur together. Using longitudinal context data collected from 106 users over 1-3 months, we showed that MobileMiner efficiently generates patterns using limited phone resources, achieving a performance improvement of 15 times over the Apriori mining algorithm, and generating overall frequent patterns in 16 minutes and detailed app usage patterns in 21 seconds. We found interesting behavior patterns for individual users and across users, ranging from calling patterns to place visitation patterns. Finally, we showed how our co-occurrence patterns can be used to improve the phone UI for launching apps or calling contacts, by predicting the next app or contact invoked based on MobileMiner patterns; we concluded with a user survey that showed the promise of our app prediction service.

In the future, we propose to explore several research directions to improve MobileMiner. We plan to explore co-occurrence patterns of events over longer time durations of the order of hours or days. To improve efficiency, we propose to evaluate approximate, incremental rule mining algorithms [9]. We plan to to systematically determine the correct frequency of running the mining algorithms based on the rate of change of user patterns and the application needs. In the future, we also plan to perform a comparison of our context prediction approach with more widely used classifier-based approaches, from the standpoints of accuracy, efficiency, usability, and ability to trade off recall-precision. Finally, we are interested in extending our work to other types of patterns such as sequential or correlation patterns to improve the potential benefits for end users.

REFERENCES

1. Aggarwal, C. C., and Yu, P. S. A new approach to online generation of association rules. *IEEE Transactions on Knowledge and Data Engineering 13*, 4 (2001), 527–540.
2. Agrawal, R., and Srikant, R. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94)*, Morgan Kaufmann (1994).
3. Aharony, N., Pan, W., Ip, C., Khayal, I., and Pentland, A. Social fmri: Investigating and shaping social mechanisms in the real world. *Pervasive and Mobile Computing 7*, 6 (2011).
4. Allen, J. F. Maintaining knowledge about temporal intervals. *Communications of the ACM 26*, 11 (1983), 832–843.
5. Android operating system. **http://www.android.com/**.
6. Azizyan, M., Constandache, I., and Roy Choudhury, R. Surroundsense: Mobile phone localization via ambience fingerprinting. In *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking (MobiCom '09)* (2009).
7. Banerjee, N., Agarwal, S., Bahl, P., Chandra, R., Wolman, A., and Corner, M. Virtual compass: Relative positioning to sense mobile social interactions. In *Proceedings of the 8th International Conference on Pervasive Computing (Pervasive '10)*, Springer-Verlag (2010).
8. Borgelt, C. Efficient implementations of apriori, eclat and fp-growth. **http://www.borgelt.net**, August 2013.
9. Cheung, D. W., Han, J., Ng, V. T., and Wong, C. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Data Engineering, 1996. Proceedings of the Twelfth International Conference on*, IEEE (1996), 106–114.
10. Samsung galaxy s4. **http://www.samsung.com/latin_en/consumer/mobile-phones/mobile-phones/smartphone/GT-I9500ZKLTPA-spec**.
11. Samsung gear. **http://www.samsung.com/us/mobile/wearable-tech**.
12. Han, J., Kamber, M., and Pei, J. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., 2011.
13. Hao, T., Xing, G., and Zhou, G. isleep: Unobtrusive sleep quality monitoring using smartphones. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys '13)*, ACM (2013).
14. Ifttt mobile recipes. **https://ifttt.com/recipes**.
15. ios 7. **https://www.apple.com/ios/what-is/**.
16. Kwapisz, J. R., Weiss, G. M., and Moore, S. A. Activity recognition using cell phone accelerometers. *SIGKDD Explorations Newsletter 12*, 2 (2011), 74–82.
17. Li, W., Han, J., and Pei, J. Cmar: accurate and efficient classification based on multiple class-association rules. In *Proceedings of IEEE International Conference on Data Mining (ICDM '01)*, IEEE (2001).
18. Lin, K., Kansal, A., Lymberopoulos, D., and Zhao, F. Energy-accuracy trade-off for continuous mobile device location. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys '10)*, ACM (2010).
19. Linux foundation. **http://collabprojects.linuxfoundation.org/**.
20. Liu, B., Jiang, Y., Sha, F., and Govindan, R. Cloud-enabled privacy-preserving collaborative learning for mobile sensing. In *Proceedings of the 10th ACM Conference on Embedded Networked Sensor Systems (SenSys '12)*, ACM (2012).
21. Liu, J., Priyantha, B., Hart, T., Ramos, H. S., Loureiro, A. A. F., and Wang, Q. Energy efficient gps sensing with cloud offloading. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems (SenSys '12)*, ACM (2012).
22. Lu, H., Pan, W., Lane, N. D., Choudhury, T., and Campbell, A. T. Soundsense: Scalable sound sensing for people-centric applications on mobile phones. In *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services (MobiSys '09)*, ACM (2009).
23. Miluzzo, E., Lane, N. D., Fodor, K., Peterson, R., Lu, H., Musolesi, M., Eisenman, S. B., Zheng, X., and Campbell, A. T. Sensing meets mobile social networks: The design, implementation and evaluation of the cenceme application. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems(SenSys '08)*, ACM (2008).
24. Monsoon power monitor. **https://www.msoon.com/LabEquipment/PowerMonitor/**.
25. Nath, S. Ace: Exploiting correlation for energy-efficient and continuous context sensing. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys '12)*, ACM (2012).
26. Parate, A., Böhmer, M., Chu, D., Ganesan, D., and Marlin, B. M. Practical prediction and prefetch for faster access to applications on mobile phones. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, ACM (2013), 275–284.
27. Shin, C., Hong, J.-H., and Dey, A. K. Understanding and prediction of mobile application usage for smart phones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing (UbiComp '12)*, ACM (2012).
28. Srinivasan, V., Moghaddam, S., Mukherji, A., Rachuri, K., Xu, C., and Tapia, E. M. On-device mining of mobile users' co-occurrence patterns. In *Proceedings of the 15th International Workshop on Mobile Computing Systems and Applications (POSTER)* (2014).

29. Survey monkey. **https://www.surveymonkey.com/**.

30. Tizen platform. **https://www.tizen.org**.

31. Welbourne, E., Wu, P., Bao, X., and Munguia-Tapia, E. Crowdsourced mobile data collection: lessons learned from a new study methodology. In *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*, ACM (2014), 2.

32. Yan, T., Chu, D., Ganesan, D., Kansal, A., and Liu, J. Fast app launching for mobile devices using predictive user context. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, ACM (2012), 113–126.

33. Yin, X., and Han, J. Cpar: Classification based on predictive association rules. In *Proceedings of the 2003 SIAM International Conference on Data Mining (SDM '03)*, SIAM (2003).

34. Zaki, M. J. Spade: An efficient algorithm for mining frequent sequences. 31–60.

35. Zou, X., Zhang, W., Li, S., and Pan, G. Prophet: What app you wish to use next. In *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication (UbiComp '13 Adjunct)*, ACM (2013).