UNIVERSITY OF BIRMINGHAM
University of Birmingham Research at Birmingham

# PrefMiner: Mining User's Preferences for Intelligent Mobile Notification Management

Mehrotra, Abhinav; Hendley, Robert; Musolesi, Mirco

*DOI:*
10.1145/2971648.2971747

*License:*
None: All rights reserved

*Document Version*
Peer reviewed version

*Citation for published version (Harvard):*
Mehrotra, A, Hendley, R & Musolesi, M 2016, PrefMiner: Mining User's Preferences for Intelligent Mobile Notification Management. in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2016)* . Association for Computing Machinery , pp. 1223-1234 , 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2016) , Heidelberg, Germany, 12/09/16. https://doi.org/10.1145/2971648.2971747

Link to publication on Research at Birmingham portal

**Publisher Rights Statement:**
© ACM, 2016. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing

**General rights**
Unless a licence is specified above, all rights (including copyright and moral rights) in this document are retained by the authors and/or the copyright holders. The express permission of the copyright holder must be obtained for any use of this material other than for purposes permitted by law.

•Users may freely distribute the URL that is used to identify this publication.
•Users may download and/or print one copy of the publication from the University of Birmingham research portal for the purpose of private study or non-commercial research.
•User may use extracts from the document in line with the concept of 'fair dealing' under the Copyright, Designs and Patents Act 1988 (?)
•Users may not further distribute the material nor use it for the purposes of commercial gain.

Where a licence is displayed above, please note the terms and conditions of the licence govern your use of this document.

When citing, please reference the published version.

**Take down policy**
While the University of Birmingham exercises care and attention in making items available there are rare occasions when an item has been uploaded in error or has been deemed to be commercially or otherwise sensitive.

If you believe that this is the case for this document, please contact UBIRA@lists.bham.ac.uk providing details and we will remove access to the work immediately and investigate.

Download date: 19. Mar. 2024

# PrefMiner: Mining User's Preferences for Intelligent Mobile Notification Management

**Abhinav Mehrotra**
University of Birmingham
University College London
United Kingdom
a.mehrotra@cs.bham.ac.uk

**Robert Hendley**
University of Birmingham
United Kingdom
r.j.hendley@cs.bham.ac.uk

**Mirco Musolesi**
University College London
United Kingdom
m.musolesi@ucl.ac.uk

## ABSTRACT

Mobile notifications are increasingly used by a variety of applications to inform users about events, news or just to send alerts and reminders to them. However, many notifications are neither useful nor relevant to users' interests and, also for this reason, they are considered disruptive and potentially annoying.

In this paper we present the design, implementation and evaluation of PrefMiner, a novel interruptibility management solution that learns users' preferences for receiving notifications based on automatic extraction of rules by mining their interaction with mobile phones. The goal is to build a system that is *intelligible* for users, i.e., not just a "black-box" solution. Rules are shown to users who might decide to accept or discard them at run-time. The design of PrefMiner is based on a large scale mobile notification dataset and its effectiveness is evaluated by means of an *in-the-wild* deployment.

### Author Keywords

Notifications; Interruptibility; Context-aware Computing.

### ACM Classification Keywords

H.1.2. Models and Principles: User/Machine Systems; H.5.2. Information Interfaces and Presentation (e.g. HCI): User Interfaces

## INTRODUCTION

Today's mobile phones are highly personal devices characterized by always-on connectivity and high-speed data processing. These affordances make it a unique platform for applications harnessing the opportunity of real-time information delivery. A variety of applications are available on the app stores that enable users to subscribe to numerous information channels and actively receive information through notifications [3, 4].

Previous studies have shown that users are willing to tolerate some interruptions from notifications, so that they do not miss any important information [24]. However, their willingness is, in a sense, exploited by mobile applications as these trigger a plethora of notifications continuously [25]. Given the potentially large number of notifications, users do not accept all of them as their receptivity relies on the content type and the sender of the messages [25, 26]. Users mostly dismiss (i.e., swipe away without clicking) notifications that are not useful or relevant to their interests [19, 33]. Some examples of such notifications are promotional emails, game invites on social networks and predictive suggestions by applications. At the same time, past studies have shown that users get annoyed by receiving irrelevant or unwanted notifications which could result in uninstalling the corresponding application [17, 33].

The above findings provide evidence that, in order to reduce the level of disruption, an interruptibility management system should not just try to deliver notifications at opportune moments but also stop notifications that are not useful, uninteresting or irrelevant for the user. However, most of the previous studies propose interruptibility management mechanisms that focus on modeling interruptibility to predict opportune moments by using context [21, 18, 29, 27] and content [25]. In this work, for the first time, we design an intelligent interruptibility management mechanism that *learns the types of information users prefer to receive via notifications in different situations*. Another important aspect is usability: in order to tackle this problem we implement a mechanism for mining association rules [9] and *make the discovered rules transparent to users so that they can check their appropriateness*.

In order to train and evaluate the proposed intelligent notification mechanism, we first exploit the datasets of real-world mobile notifications collected during the My Phone and Me study [26]. We construct the association rules by using the combinations of notification titles and context modalities including activity, time and location. Through an extensive evaluation, we show that by using notification titles and the user's location we can predict the notifications that will be dismissed by the user with a precision greater than 91%. Moreover, we show that the user's activity and hour of the day do not contribute to the improvement of the prediction accuracy.

We then funneled our findings into a practical implementation of an interruption mechanism for mobile devices – *MyPref*, an intelligent notification library for the Android OS. MyPref enables an application to discover personalized rules for the user's preferences and predict their receptivity to notifications

Paste the appropriate copyright statement here. ACM now supports three different copyright statements:
• ACM copyright: ACM holds the copyright on the work. This is the historical approach.
• License: The author(s) retain copyright, but ACM receives an exclusive publication license.
• Open Access: The author(s) wish to pay for the work to be open access. The additional fee must be paid to ACM.
This text field is large enough to hold the appropriate release statement assuming it is single spaced.
Every submission will be assigned their own unique DOI string to be included here.

containing different types of information. The expressive API of the library allows a developer to configure the prediction settings including the features used for mining rules, making it also extensible and generalizable. Moreover, through an extensive evaluation of memory, battery and time overheads, we show that MyPref is a light-weight and resource efficient library that is capable of mining 1500 notification samples in a minute by consuming only 11.737 mAH battery charge.

We present the design and implementation of PrefMiner, an interruptibility management application that is built on top of the MyPref library. To achieve users' trust the application shows the discovered rules to them and asks for their consent for each rule to activate it.

Finally, unlike previous interruptibility studies, we do not restrict ourselves to evaluating the interruptibility management mechanism on the collected data or to manage our own generated notifications. Instead, in order to evaluate PrefMiner, we conducted an *in-the-wild* 15-day experiment. We show that with 16 subjects PrefMiner suggested 179 rules and 56.98% of these rules were accepted. During the study period the app filtered notifications with a recall of 45.81% and a precision of 100% as all filtering rules were accepted by the users.

## MOTIVATION OF KEY DESIGN CHOICES

In this section we discuss the key motivations and design choices of our solution, namely (i) the choice of learning users' preferences with respect to different types of information delivered through notifications rather than modelling their interruptibility; (ii) the choice of implementing an algorithm based on association rules instead of alternative machine learning algorithms.

### Why Mine User's Preferences for Different Types of Information?

The key objective of an interruptibility management system is to deliver the right information at the right time. Most of the previous interruptibility studies have proposed different approaches to model interruptibility for predicting opportune moments [21, 18, 29, 25]. However, these studies do not suggest what to do with notifications that arrive at inopportune moments. Should we defer them? Or should we completely dismiss them? A previous study [22] shows that notifications can be deferred for a few minutes to reduce disruption in the desktop environment. However, this approach could not be applied in mobile settings because the user's context might change continuously and the variety of notifications received on mobile phones is much wider than desktop notifications.

In a previous study [13], Clark suggested that users' negative response to an interruption can be of two types: (a) acknowledge it and agree to handle it later; (b) decline it (explicitly refusing to handle it). Based on Clark's suggestions *we hypothesize that an interruptibility management mechanism should take an orthogonal but equally important approach by learning the different types of interruptions that users explicitly refuse by dismissing notifications*. In this way such mechanism can identify the notifications that are not useful for the users and stop the operating system from triggering alerts for these types of notification. Moreover, our hypothesis is inline with the findings of Fischer et al. [19, 33], i.e., users' receptivity relies on the usefulness and their interest in the delivered information. We would like to stress again that this is an orthogonal mechanism that can be used in conjunction with others, for example for deferring the notifications until the right time/context.

### Why Mine Association Rules Over other Types of Machine Learning Models?

One of the major issues in designing interruptibility management systems is related to their testing and evaluation. In fact, if the notification mechanisms are not correct, they might lead to rejecting or deferring important notifications. Therefore, in order to build *usable* systems, we have to involve users to adjust the interruptibility management mechanisms, in order to achieve the goal of reducing interruptions without compromising the reception of any useful and important information.

The interruptibility management mechanisms proposed in the previous studies rely on machine learning models for making predictions that might be difficult to understand and translate directly into human-readable rules [21, 29, 25]. These models are good for learning quickly with a high accuracy but it is nearly impossible for users to understand these models and provide their feedback. Therefore, *we rely on mining association rules that can be easily understood by users as compared to other prediction models*. This allows us to get feedback from users about the rules that should not be used for stopping notifications on their phones. It is worth noting that a simple machine learning technique such as the decision tree [34] is likely to find a few more rules than the association rules, however, the decision tree-based rules mostly have low reliability because they refer to very small sets of data instances [28].

## MINING USER PREFERENCES

In this section we discuss how we extract notification rules by mining association rules based on different combinations of notification titles and context modalities including activity, time and location.

### Removing Reminder Notifications

The first step consists of identifying a particular class of notifications that are always dismissed but they should be shown in any case to users. As discussed earlier, notifications are dismissed if they are not found to be useful or relevant to the user's interest [19, 33]. However, some notifications are dismissed because they do not require any further action from the user. These notifications should not be automatically filtered, since they might be relevant for users, even if they are always dismissed. We refer to such notifications as *reminder notifications* in the rest of the paper. Alarm, calendar event and battery status notifications are some common examples of reminder notifications.

More formally, in order to define reminder notifications, we introduce a simple definition of *click rate (CR)*:

$$\mathrm{CR} = \frac{\text{Number of accepted notifications}}{\text{Total number of notifications}} * 100 \qquad (1)$$

If an application's click rate (CR) is zero then all notifications from that application are treated as reminder notifications.

#### Notification Classification

In order to model users' preferences with respect to receiving different types of information, we categorize each notification based on the information contained in it. A recent study proposes an approach for modelling interruptibility by using information type, social circle and context information [25]. The authors of [25] assume that all notifications triggered by an application are of the same type and categorize them by using the type of application that triggered them. In other words, they classify notifications at an abstract level, e.g., chat, email, systems and so on.

Instead, we do not limit ourselves to categorize a notification based on the type of application that triggered it because an application can generate notifications that contain different types of information. A user might be interested to receive some but not all types of notifications triggered by a specific application. For example, a Facebook notification about a new post on a user's timeline might not be considered as disruptive as a game invite notification. In other words, users would not want an interruptibility management system to completely stop Facebook notifications, but only those that are annoying, such as game invites.

Therefore, in order to classify notifications, we perform clustering by considering their titles by means of DBSCAN [16], a density based algorithm[1]. A notification title is a short sentence that gives a glimpse of the information contained in it. The following are some examples of notification titles: *"Sign in to a Wi-Fi network"*, *"Time to Work"*, *"Today is Alice's birthday"*. It is worth noting that in some cases the notification title contains the sender name along with other text (such as *"Alice commented on your post"*) or merely the sender name (such as *"Alice"*). Generally, the sender name is attached to the titles of notifications triggered by chat and online social networking applications.

The clustering of notifications is carried out through the following steps: i) cleaning notification titles; ii) constructing a classifier; and iii) clustering notifications. We discuss the details of each step in detail below.

##### *Cleaning Notification Titles*

Notification titles are short sentences phrased in a way that they are easily understood by users. The most important step for analyzing these titles is to first clean them in order to remove the non-informative data.

To analyze the notification titles we follow the standard process of cleaning text in the following way:

(i) *Conversion of the the text to lower-case*: this ensures that lower-case and upper-case versions of the same word are considered the same.

(ii) *Removal of punctuation and numbers*: these elements of the text do not contain useful information for the classification task, but at the same time they might influence it.

(iii) *Removal of stop words*: stop words are the common words (such as *'a'*, *'the'*, *'is'* and *'are'*) that generally have quite high frequency in the text. We remove them to ensure that they do not affect *content-bearing keywords* in the clustering algorithm [12].

(iv) *Removal of the sender and application names*: we remove sender and application names because they can lead the classifier to cluster notifications based on the sender or application names. Therefore, for each notification we remove (if present) the name of the application by which it is triggered. Moreover, to remove sender names we find and remove all the words that are not present in the english dictionary that comes with `qdapDictionaries` package [5]. In case of chat notifications that contain only names, we do not remove any word and thus allow the algorithm to classify chat notifications based on the sender names.

(v) *Stemming of words*: it is a standardization method to avoid having multiple versions of words referring to the same concept by reducing a word down to its root. For instance, the words *'comment'*, *'commented'*, *'comments'* and *'commenting'* are all stemmed to *'comment'*).

##### *Constructing a Classifier*

In order to train the classifier, we use a bag of words approach and create a *Document-Term Matrix (DTM)* – a matrix that describes the frequency of terms (i.e., words) that occur in a collection of documents (i.e., notification titles in our case). In a DTM, rows correspond to documents in the collection and each term is associated to a column.

To prevent the problem of overfitting the classifier [35], we compute the term frequency ($TF$) and remove the terms that have a $TF$ lower than $TF_{threshold}$. The $TF$ and $TF_{threshold}$ are defined as following:

$$\text{TF} = \frac{\text{Number of notifications in which the word occurs}}{\text{Total number of notifications}} \tag{2}$$

$$\text{TF}_{\text{threshold}} = \frac{\text{Number of participation days}}{\text{N * Total number of notifications}} \tag{3}$$

According to the above equation the value of $\text{TF}_{\text{threshold}}$ ensures that at least one notification containing the term is triggered in $N$ days.

Finally, a DBSCAN-based classifier is constructed by using the above DTM. In order to cluster the data the classifier requires two parameters as inputs: (i) $MP$, defined as the minimum number of points required to form a dense region and (ii) $\varepsilon$, defined as the maximum difference (i.e., number of non-matching words) between the notification titles of a cluster.

##### *Clustering Notifications*

Since the notification titles are relatively short-length sentences, it is possible that notifications from different applications contain similar words that can lead the classifier to

[1]We are aware that notification clustering could be improved by using the entire notification content (i.e., including also the header of notifications). However, due to privacy concerns the notification headers were not recorded in the dataset we used for our evaluation [26].

cluster them together. In order to prevent this problem, for each application we create a separate classifier by using the notifications generated by that application. For example, if there are 15 notifications from 3 applications, we create 3 clustering models and each model is trained using notifications of separate applications. Finally, when the classifiers are constructed, we predict the classes of all notifications.

### Constructing Association Rules

In order to discover rules about the user's preferences for receiving notifications, we use the AIS algorithm [9] – a method for mining data to discover statistical relationships between variables. The algorithm scans the data to find the frequent item sets and computes their support value (discussed later in this section). Finally, it filters out the list of item sets whose support value is greater than the given support threshold.

An association rule that is represented as $X \rightarrow Y$, where $X$ is defined as the antecedent and $Y$ as the consequent. The algorithm generates rules with the consequent containing only one item. This means that rules can be in the form of $X_1 \cup X_2 \rightarrow Y$ but not in the form of $X \rightarrow Y_1 \cup Y_2$.

To better understand the concept of association rules let us consider an example where the user: (i) always dismiss Twitter notifications for who to follow; (ii) accepts Facebook birthday reminder notifications only in the morning while she is at home; (iii) does not accept WhatsApp notifications from Alice while at work. Assuming that notifications about the Twitter suggestion, Facebook birthday reminder and WhatsApp from Alice are classified in the classes $N_1$, $N_2$ and $N_3$ respectively, the following association rules would represent the user's preferences in this case:
$\{N_1\} \rightarrow \{Dismiss\}$
$\{N_2, Home, Morning\} \rightarrow \{Accept\}$
$\{N_2, Home, Afternoon\} \rightarrow \{Dismiss\}$
$\{N_2, Home, Evening\} \rightarrow \{Dismiss\}$
$\{N_2, Home, Night\} \rightarrow \{Dismiss\}$
$\{N_2, Work\} \rightarrow \{Dismiss\}$
$\{N_2, Other\} \rightarrow \{Dismiss\}$
$\{N_3, Home\} \rightarrow \{Accept\}$
$\{N_3, Other\} \rightarrow \{Accept\}$
$\{N_3, Work\} \rightarrow \{Dismiss\}$

For an association rule $X \rightarrow Y$, we define the two parameters:

- *Support*: the ratio between the number of times $X$ and $Y$ co-occur and the number of data-instances present in the given data. It can be represented as the joint probability of $X$ and $Y$: $P(X,Y)$;

- *Confidence*: the ratio between the number of times $Y$ co-occurs with $X$ and the number of times $X$ occurs in the given data. It can be represented as the conditional probability of $X$ and $Y$: $P(Y \mid X)$.

An association rule is created only when it has the minimum support ($S_{min}$) and confidence ($C_{min}$). It is worth noting that decreasing the values of either support or confidence could result in discovering more rules [9].

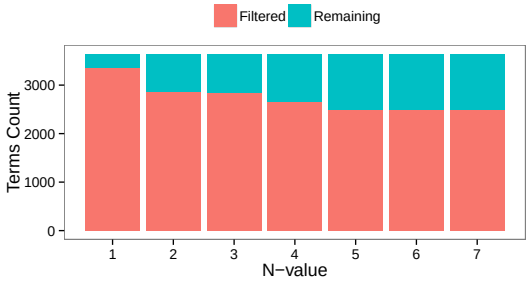



**Figure 1. Filtered and remaining terms for notification clustering with different values of $N$ in $TF_{threshold}$.**

## EVALUATION OF THE RULE-BASED MECHANISM

In this section we discuss the implementation and evaluation of the mechanism for mining individual-based association rules about users' preferences.

### Dataset and Evaluation Settings

In order to evaluate the proposed solution, we use the data collected by means of smartphones during the My Phone and Me study [26]. From the original dataset, we took a subset of data considering only the users who participated for at least 14 days. We also removed the notifications which were not mapped with the user's activity, location and arrival time. Moreover, the dataset contains an attribute to identify whether a notification is clicked, dismissed or handled on another device. We did not consider notifications that were handled on another device. Consequently, the final dataset used in our analysis comprises 11,185 notifications from 18 users.

We evaluate the discovered rules for predicting their response to notifications by using a $k$-fold cross validation approach with the value of $k$ as 10.[2] We divide the set of notifications belonging to each user into a training set and a test set, where the training set contains 90% of the data and the rest is considered as the test set.

### Defining Configurations

As discussed earlier, we use the $TF_{threshold}$ to prevent the problem of overfitting the DBSCAN-based classifier with sparse (i.e., infrequent) terms. In order to find an optimal value of $TF_{Threshold}$ we create a DTM for all notification titles in the dataset and compute the number of terms filtered by setting $TF_{Threshold}$ with different values of $N \in [1,7]$. Here, we do not consider values of $N$ greater than 7 because we believe it would not be useful to include notification terms that do not arrive at least once in a week.

As shown in Figure 1 there are 3642 unique terms in the dataset. By setting $N = 1$, there are chances of underfitting the model because 3371 terms are removed and only 271 terms remain in the DTM. Since there is not much difference in the number of filtered terms with $N \in [2,7]$, we use $N = 2$ for our analysis, which also ensures that each term is seen by the user at least once in two days.

[2]We compute prediction results separately for each user and aggregate them by computing the mean and the standard-error with 95% confidence limits.

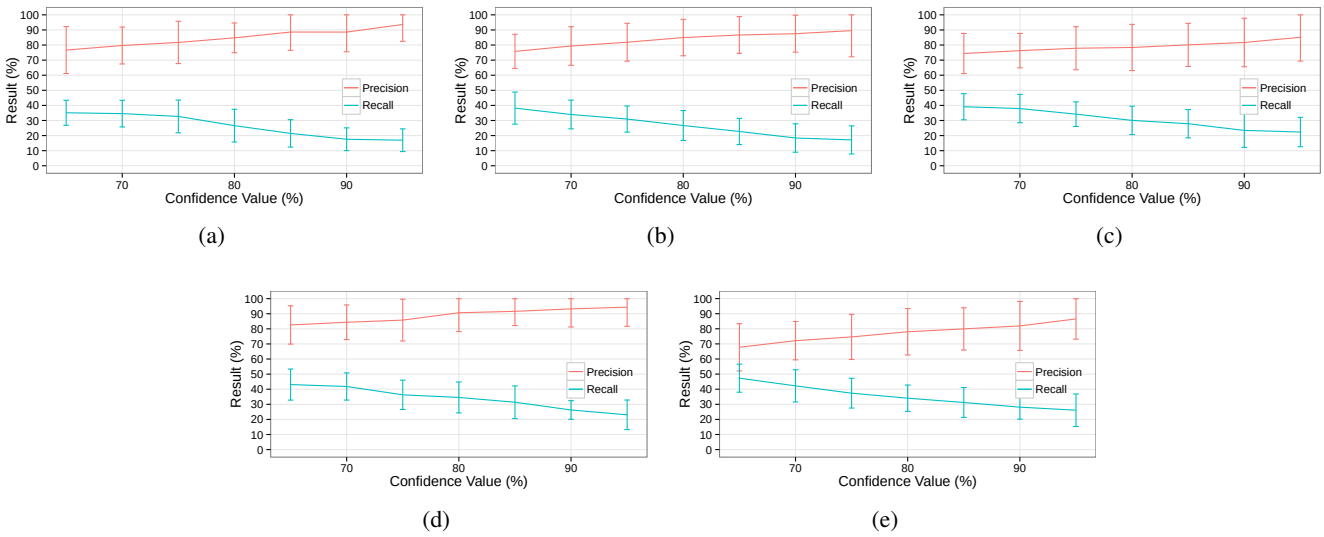



**Figure 2. Prediction results for association rules discovered by using notification response along with: (a) AR1: notification type; (b) AR2: notification type and activity; (c) AR3: notification type and arrival time; (d) AR4: notification type and location; (e) AR5: notification type, activity, arrival time and location.**

In order to satisfy the above constraint, for each user we set the value of $MP$ to $D/2$ ($D$ indicates the number of participation days). This ensures that there are at least $D/2$ notifications in each cluster. In other words, notifications of each cluster arrive at least once in two days. Similarly, for discovering association rules we set $S_{min}$ (i.e., minimum support value) equals to $D_{participate}/(2 * N_{count})$, which ensures that the notification interaction pattern covered in each rule has occurred at least once in two days. Here, $D_{participate}$ indicates the number of participation days of the user and $N_{count}$ refers to the total number of notifications collected for that user. Moreover, we set $\varepsilon$ to 1 so that each notification title will have at least $N-1$ ($N$ refers to the number of words in a title) words similar to other notification titles in its cluster.

### Feature Selection

In order to discover association rules about the user's preferences we rely on the following features:

(i) *notification response*: the user's response (i.e., click or dismiss) to a notification;

(ii) *notification type*: the identifier of the cluster to which the notification belongs;

(iii) *arrival time*: the arrival time of the notification considering four time slots – morning (6-12), afternoon (12-16), evening (16-20) and night (20-24 and 0-6);

(iv) *activity*: the user's physical activity (includes still, walking, running, biking and in vehicle) when the notification arrived;

(v) *location*: the user's location when the notification arrived;

(vi) *response*: the user's response (acceptance or dismissal) to the notification.

By using different combinations of these features we construct the association rules according to the following five approaches:

1. **AR1**: by using notification response with notification type;
2. **AR2**: by using notification response with notification type and activity;
3. **AR3**: by using notification response with notification type and arrival time;
4. **AR4**: by using notification response with notification type and location;
5. **AR5**: by using notification response with notification type, activity, arrival time and location;

For all approaches we restrict the consequent to contain only the notification response and the antecedent is restricted to never contain the notification response. We introduce this constraint because we are only interested to predict the acceptance of a notification, therefore other items in the consequent would be of no use and just add extra computational load.

### Prediction Results

In this section we present the accuracy of the association rules discovered with different values of $C_{min}$ for all five approach. In order to assess the discovered association rules, we compare the predicted response with the actual response (i.e., the ground truth) and compute the accuracy in terms of:

- *Recall*: ratio between the number of notifications that are correctly predicted as dismissed and the total number of notifications that are actually dismissed.
- *Precision*: ratio between the number of notifications that are correctly predicted as dismissed and the total number of notifications that are predicted (both correctly and incorrectly) as dismissed.

In Figure 2 we present the prediction results for the association rules constructed by using all five approaches. The results show that increasing the confidence of association rules decreases the recall but improves the precision. This implies that

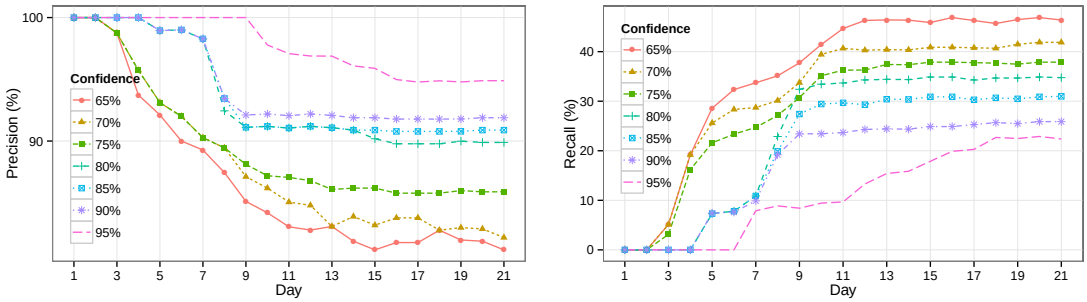



**Figure 3. Prediction results for association rules (with notification title and location) using the online learning approach.**

by increasing the confidence a fewer but more reliable rules are discovered.

The association rules constructed with approaches *AR1* and *AR2* do not show a significant difference. The recall of *AR2* is slightly higher when the confidence is below 70%, but the precision drops as well, which means some extra but unreliable rules are discovered. On the other hand, the association rules constructed with *AR3* achieves better recall than *AR1* and *AR2* but its precision consistently remains lower (i.e., under 85%) as compared to other approaches. This implies that there is no significant contribution of activity and arrival time in terms of predicting user's preferences for receiving notifications.

The approach *AR4* (i.e., the association rules that are constructed by using the notification response, type and location) performs better that other approaches in terms of both recall and precision. Its recall goes up to around 43% without dropping the precision below 79%. Even by combining all features together in the approach *AR5*, the results do not improve. As compared to approach *AR4*, there is a negligible increment in the recall of the approach *AR5* with a big drop in its precision. *Consequently, our results provide evidence that the user's preference for receiving notifications does not depend on the activity and arrival time, but on the type of information it contains and the location of the user.*

It is worth noting that the high standard-error in the results demonstrates that our mechanism does not work consistently for all users. It is due to the fact that users have different preferences for receiving notifications. Moreover, the upper-limit of the standard-error for precision close to 100% demonstrate that for some users our mechanism would be able to very accurately predict the notifications that are not interesting or useful for them.

Another interesting observation is that the recall of approach *AR3* is always higher than that of *AR1* and *AR2*, but its precision is instead always lower. We believe that *AR3* attains higher recall because users follow circadian rhythms and during different time periods of a day they are present at certain locations (e.g., at work in the afternoon and at home during night) [15]. Therefore, in approach *AR3* rules are essentially associated to their behavior at certain locations rather than the time of notifications' arrival. Moreover, the precision might be affected by the fact that user's mobility pattern might vary on different days.

### Optimizing the System for High Precision

The key requirement for deploying an interruptibility management mechanism *in-the-wild* is that it should never stop/defer useful notifications. Therefore, while designing it we should aim to have fewer false-negatives (i.e., incorrectly prediction of a notification as non-interesting for the user) that could be achieved by ensuring that the precision remains close to 100%. However, we could consider a precision of around 90% as acceptable because it might be possible that a couple of notifications have been clicked by mistake or to kill time when the user was bored.

At the same time, the interruptibility management mechanism should also achieve a significant value of the recall in order to prove its efficacy of filtering notifications that are not useful or relevant to the user's interest. We could not obtain a high recall value because not all dismissed notifications are non-useful. Instead, some notifications are dismissed because they do not require any further actions. Though we have already filtered out reminder notifications before the analysis, there could be notifications from other applications which are useful, but do not require any action, such as a chat message *"See you at 7pm"* and a confirmation email *"Ok, bye!"*.

We observe that, by using the confidence of 70%, the approach *AR4* is able to achieve around 42% recall with a precision of 84%. However, the drop of 16% precision value is a risk that the mechanism could predict some false-negatives. So, we should compromise by having around 35% recall with a precision of more than 90% at a confidence of 80%. Here, even the 35% recall implies that we are able to filter a big portion of non useful notifications and thus reduce disruption.

## ONLINE LEARNING

In the previous section we evaluated our mechanism by using the batch learning method in which the association rules are mined by using static data. However, in a real world scenario such training data is not initially available and therefore, association rules cannot be discovered until a sufficient amount of data is available.

In order to address this issue, we can use an online learning method in which the association rules are periodically mined

from the data that is collected gradually. Even if this solution does not remove the problem of initial bootstrapping completely[3], the prediction accuracy improves gradually as more and more training data becomes available. Moreover, by using an online learning method a system can adapt itself according to the potential changes in user's behavior over time.

In order to evaluate this method, we iteratively construct association rules with all the notifications collected by the end of each day and evaluate these rules by using notifications of the following day. For example, on day $N$ a model is built by using notifications from day 1 to $N-1$. Moreover, similar to the evaluation by using batch learning method we configure $S_{min}$ as $(N-1)/(2*N_{\text{count}})$, where $N_{\text{count}}$ indicates the total number of notifications collected till day $N-1$. This ensues that the notification interaction pattern covered in each rule has occurred at least once in two days or at least $(N-1)/2$ times in $N-1$ days.

Figure 3 shows the prediction results for the individualized association rules iteratively constructed on each day with different values of $C_{min}$ by using the online learning method. Rules with $C_{min}$ as 80 and below start filtering notifications from the $3^{rd}$ day and by the $7^{th}$ day they achieve the recall of 25-35% and precision around 90%. Instead, by the $7^{th}$ day other rules could achieve the recall of 10% and precision above 90%.

Interestingly, rules with $C_{min}$ as 95% never become stable. This can be due to the fact that these rules filter notifications with high reliability which can be confirmed by their precision always remaining over 90%. On the other hand, rules with $C_{min}$ as 65-70% and 80-90% become stable by the $12^{th}$ and $9^{th}$ day respectively.

It is worth noting that given the duration of the dataset, it is not possible to evaluate the adaptation of the algorithm and confirm if the rules maintain their stability over time. It might be possible that users diverge from their notification interaction behaviour and the system needs to adapt accordingly. Therefore, we envisage that the system should use a sliding window approach, i.e., the algorithm should be trained on the last $L$ days in order to be able to adapt to changes in user's behavior.

## MYPREF LIBRARY

### Overview

Starting from the mechanisms and evaluation we presented in the previous section, we implemented the MyPref library – an intelligent interruptibility management library that can predict the type of notifications that users would prefer to receive on their mobile phones in specific contexts. The library learns the user's preferences for receiving notifications by mining association rules with the notification and context data that is supplied to it and predicts the user's receptivity to the subsequent notifications. The MyPref library is implemented for the Android OS and released as an open source project [4]. The goal is to provide developers with a practical generic tool for intelligent rule-based notifications that can be integrated in any application, hiding at the same time the complexity related to the prediction mechanisms.

The MyPref library abstracts the functionalities of the proposed interruptibility mechanism through a set of intuitive API primitives. The abstractions include clustering of notifications, mining association rules and predicting the acceptance of notifications. The library relies on Weka for Android [7] and the Snowball stemming library [6] for clustering notifications locally on the phone. Since the computation is performed locally, the library also preserves user's privacy since no data is transmitted to a back-end server. It offers flexibility to developers by allowing them to define notification clustering configurations (i.e., $TF_{Threshold}$, $\varepsilon$ and $MP$) as well as the rule mining configurations (i.e., support, confidence and features to be used for mining rules).

Finally, the library learns the user's preferences for receiving notifications and returns the discovered rules as output. In order to enable an overlying application to facilitate the transparency of the prediction mechanism to the users, the library makes the rules human understandable by replacing each notification type with the most frequent[5] words of the relevant notification cluster (we refer to these words as *keywords* in the rest of the paper).

For instance, let us consider some examples of hypothetic *keywords* in notification clusters: the keywords from the cluster of Facebook's birthday reminder notifications (such as *"Today is Alice's birthday."* and *"Alice and Chris have birthdays today. Help them have a great day!"*) would be *"today"* and *"birthday"*. The keyword from the cluster of Google Play Store's app update notifications (such as *"2 applications updated."* and *"3 updates available."*) would be *"update"*. The keywords from the cluster of systems's WiFi availability notifications (such as *"Wi-Fi networks available"* and *"Verizon Wi-Fi available"*) would be *"Wi-Fi"* and *"available"*.

The overlying application can present these rules to users in order to ask for their consent and use the rules that are accepted by them to filter notifications. Later in the section, we will show a potential approach to take the consent of users for using the predicted rules in an application based on the MyPref Library.

It is worth noting that producing rules with keywords instead of cluster identifiers would reduce the computation time. Indeed, this requires the clustering of notifications in order to find their cluster identifiers and then predict the response by using the rules. This might be fine for applications that are predicting the acceptance for their own notifications. However, if an application is managing notifications from third-party application then the prediction process should be very quick so that, if required, we can cancel a notification before it alerts the user (via sound, vibration or LED light).

[3]A possible alternative for bootstrapping the system is to crowdsource a set of popular rules that are selected for example by certain population groups with similar characteristics, for example location or other demographics characteristics. This is outside the scope of the present work; we plan to investigate these aspects as part of our future research agenda.

[4]https://github.com/AbhinavMehrotra/PrefMiner

[5]We consider the most frequent after removing all the stop words and stemming the remaining words.

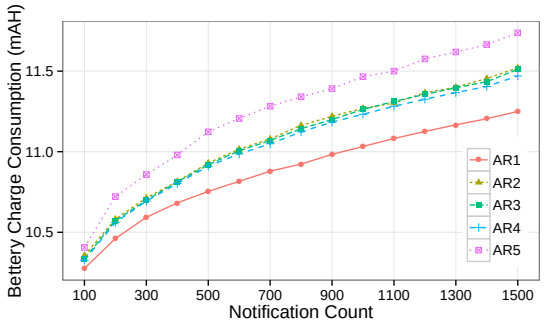



**Figure 4. Battery charge consumed for mining rules using different approaches with different number of notifications as an input.**

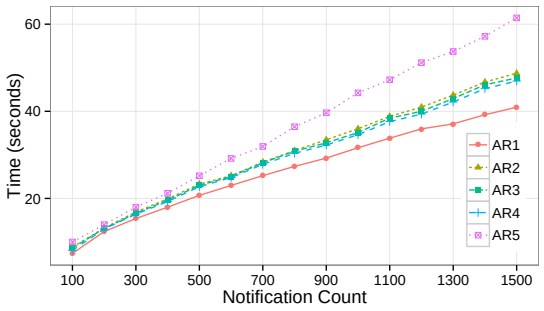



**Figure 5. Time taken for mining rules using different approaches with a varying number of notifications in input.**

## Evaluation of the MyPref Library

In this section we quantify the performance of the MyPref library. We use a Nexus 6 phone with 3 GB of RAM and a quad-core 2.7GHz Krait 450 CPU, running a clean slate Android 5.0 (Lollipop) OS for the performance evaluation of the library.

### *Source Code and Memory Footprint*

MyPref is implemented as a light-weight Android library consisting of 15 Java classes built with 4077 lines of code. We developed a stub application on top of the library to evaluate the memory footprint that accounts to 7.559 MB (including the Weka and Snowball stemming libraries). We used third-party measurement tools, namely Count Lines of Code [2] and Android Dalvik Debug Monitor Server [1], to evaluate the source code and memory footprint of the library.

### *Energy Consumption*

A fundamental aspect in designing this class of libraries is the energy usage and its impact on the phone's battery life. We analyze the energy consumption of the library by varying the amount of data in input. We characterize the battery charge consumption for mining association rules for all approaches (*AR1*, *AR2*, *AR3*, *AR4* and *AR5*). We use Power-Tutor [36] for taking the battery measurements.

As shown in Figure 4, the battery consumption increases almost linearly as the amount of data used for mining the rules increases. However, as the number of features increases the energy consumption increases but not dramatically. This implies that most of the energy is consumed for clustering the notification types and less energy is required for mining the rules.

Moreover, the battery consumption slightly varies for approaches *AR2*, *AR3* and *AR4* even when they have the same number of features used. The reason for this difference could be that these approaches use features which have different number of classes. For instance, location has three classes: *home*, *work* and *other*, whereas activity has five classes: *still*, *walking*, *running*, *biking* and *in vehicle*. Increase in the number of classes in a feature would require more iterations for discovering rules and thus consume more battery.

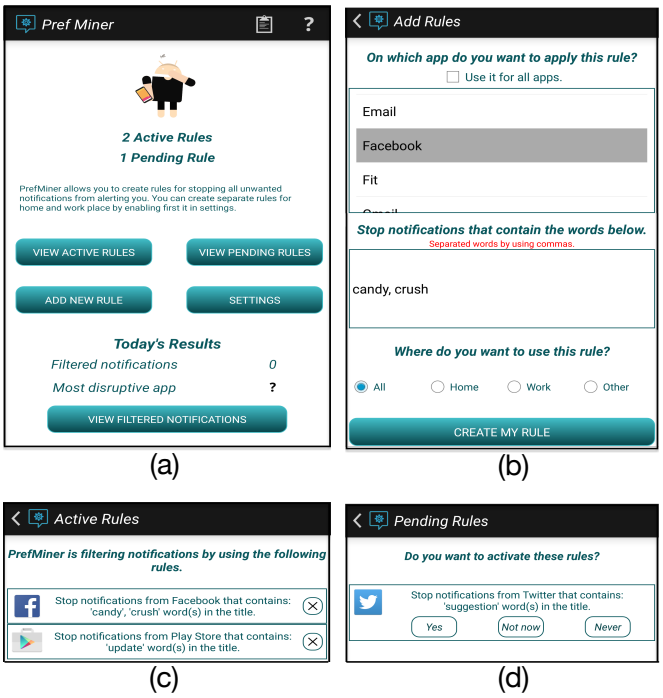



**Figure 6. PrefMiner application: (a) main screen, (b) self-rule creation, (c) active rules, (d) pending rules.**

### *Time Complexity*

We compute the time required for mining association rules by using all five approaches for different amounts of training data. The time complexity of the algorithm is $\mathcal{O}(N_{count})$.

As shown in Figure 5 and as expected, the time complexity linearly increases as the amount of data increases. Quite interestingly, the difference in terms of time required for the computation for the various approaches taken into consideration increases as the number of notifications increases. Moreover, it also increases as the number of features used gets larger. For instance, the library takes not more than 61 seconds for mining association rules from 1500 notifications for any approach.

## IN-THE-WILD EVALUATION

In this section we present an in-the-wild evaluation of PrefMiner (see Figure 6), a mobile application that is able learn the user's preference for receiving different types of no-

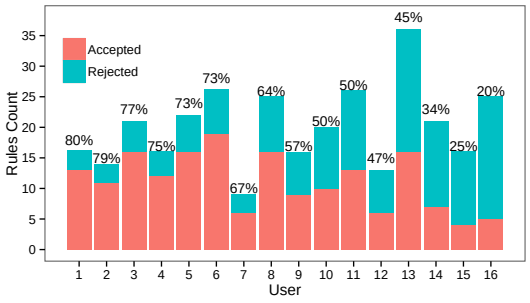



**Figure 7. Users response to the rules suggested by the PrefMiner application.**

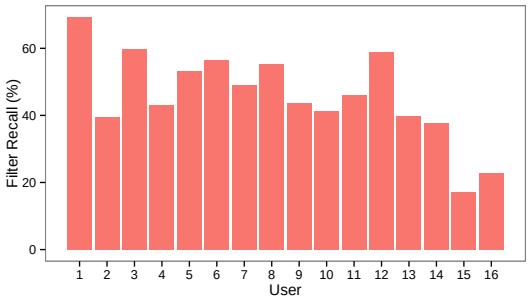



**Figure 8. Performance of PrefMiner in filtering notifications.**

| Question | Average Response |
|---|---|
| Q1. I found the app useful for learning my preferences to filter notifications. | 4.25 |
| Q2. The app filtered most of the notifications that I didn't want to receive. | 4.33 |
| Q3. The app incorrectly filtered notifications that I wanted to receive. | 1.58 |

**Table 1. Exit questionnaire with the average response from users.**

tifications and filtering out the notifications that are not useful, uninteresting and irrelevant to the user.

The PrefMiner application is built on top of the MyPref library discussed and evaluated in the previous section and uses the notification type and location data (i.e., the approach *AR4* as discussed above) for mining the association rules. The application continuously collects the notification data and binds the user's current location to each data instance. The rules are constructed every day when the phone is in charging mode and not in use so that the application does not directly affect users' mobile experience.

As shown in Figure 6.a the newly discovered rules are presented to the users in a human-readable format to get their consent. To convert a rule into a human-readable format, we use the application name, the notification cluster identifier (i.e., the keywords provided by the library as a replacement for the notification type) and location. Some possible examples of such rules are the following: *"Stop notifications from Facebook that contain 'candy' and 'crush' words in the title."* and *"Stop notifications from WhatsApp that contain 'Alice' words in the title and arrive at WORK."*

Users can accept the rules, which they think are correctly discovered, for filtering out notifications on their phones, by clicking *"Yes"*. If the user clicks on "Never" for a rule, the applications stores that rule as a blacklisted rule and never shows it again in the future. To ensure this, after every rule mining process, the application removes all blacklisted rules from the newly discovered rules. Moreover, if the user clicks on *"Not Now"*, the rule is re-proposed during the next iteration of the mining process. Note that the process of rule mining is performed once a day when the user plugs-in the phone for charging the battery to ensure that it does not hinder the user's mobile interaction experience. Finally, once a rule is accepted by a user, it becomes active (see Figure 6.c) and the application starts filtering out all the subsequent notifications according to the rules that are currently active.

### Deployment of PrefMiner

The PrefMiner application was published on the Google Play Store and advertised through social media and other channels in our University. We ran the study for the duration of 15 days followed by an exit questionnaire (see Table 1). Overall, 18 people participated in the study without any monetary incentive. However, one user did not answer any questions about the suggested rules and another user dismissed all the rules. Therefore, we considered only the remaining 16 users for evaluating the application.

To make the application interesting for the users we allowed them to manually create their own rules. As shown in Figure 6.b, a manual rule can be created by defining the application (selected from a list of installed applications), keywords and location. In order to ensure that the manual rules do not affect our experiment we restrict the users from manually create any rule during the period of the study (i.e., the first 15 days after the installation of the application). Note that, for privacy reasons, the user has to give explicit permission after the installation as required by the Android operating system to allow PrefMiner to manage notifications. Moreover, to ensure that the user is aware of data collection, the application also shows a detailed user consent form.

### In-the-wild Evaluation Results

During the study PrefMiner suggested 179 rules to the participants out of which 102 rules (i.e., 56.98%) were accepted. Figure 7 shows the count for accepted and dismissed rules for each user that are sorted according to their rule acceptance percentage. The graph shows that there are some users who accepted most of the rules and some who accepted only a few rules. Overall, around 70% of the users accepted 50% (and above) of the suggested rules.

During the study the application also collected the notifications that arrived on the user's phone. Our results show that each day a user receives around 71 notifications (with the standard deviation of 22 notifications). We compute the *filter recall* as percentage of filtered notifications over the sum of the notifications dismissed by the users and the filtered ones (i.e., all the notifications automatically or "manually" filtered). The results show that the filter recall achieved by the accepted rules is 45.81%. More specifically, the average number of

notifications that are successfully been filtered everyday is 12 (with the standard deviation as 8). In Figure 8 we show the filter recall of each user. For some users PrefMiner was able to identify around 60% of the notifications that users did not want to receive. On the other hand, the filter recall for some users was fairly low (approx. 20%). However, it is worth noting that the system is optimized for high precision so that it does not filter out any important notifications. Moreover, everyday our system considered around 7 notifications (with the standard deviation as 6) as reminder notifications for each user.

*Exit Questionnaire*

At the end of the 15-day study, the application asked users to fill in an exit questionnaire (shown in Table 1) that can be used to quantify the usefulness and accuracy of PrefMiner according to the users. Users were allowed to register their response on a 5 point Likert scale (1:Strongly disagree - 5:Strongly agree). Out of the 16 participants only 12 participants registered their response to the questionnaire. Table 1 lists the average response to the three questions provided by the participants. The results demonstrate that users found the application useful for filtering notifications and according to them it filtered most of the unwanted notifications correctly.

## DISCUSSION AND FUTURE WORK

In this paper we have presented an interruptibility management mechanism that learns the user's preference and accordingly manages the subsequent notifications. To the best of our knowledge, this is the first study based on both the analysis of a real-world notification content along with context and the *in-the-wild* deployment of a customizable notification system. We believe that our approach for making prediction techniques transparent to the users helps an interruptibility management system to build users' trust since it highly reduces the risk of stopping any important notification. It still reduces the burden from users by stopping 45.81% of notifications that are not considered useful or important and thus minimizes the perceived disruption.

However, the proposed interruptibility management mechanism has some minor limitations. Firstly, it does not focus on deferring notifications at inopportune moments. Instead, it aims to stop unnecessary notifications from alerting the user and thus reduce the overall disruption. As a future plan, we plan to design and evaluate our mechanism for learning the moments at which notifications should be deferred and for how long. Secondly, our mechanism is not able to accurately detect reminder notifications. Instead, it finds only the applications for which all notifications are dismissed and labels them as reminder applications and reminder notifications. We believe that if we can accurately identify and remove all reminder notifications, our interruptibility management mechanism would be able to discover more accurate rules and thus the overall accuracy can be improved. This might be based on the manual annotation of certain type of notifications, such as standard Android battery warning messages.

Another limitation is that we could not infer the user response to notifications that are handled from other devices. Thus, we have to discard such notifications. This reduces the overall amount of data collected for mining rules. We believe that by being able to detect the user response for such notifications, our mechanism can start making stable prediction is fewer days and might also discover some interesting rules across devices. For this reason, another extension of this study is the design of a rule-based management system across devices.

Finally, the current implementation of the MyPref library can only support mobile phones that are configured to use the English language. This is because these phones receive the notification titles in English and the current implementation of our library could classify only the text in this language. We plan to evolve our library's capability to support other languages in the future.

## RELATED WORK

The area of mobile interruptibility has received an increasing attention in the past years. Previous studies have explored various interesting aspects of the problem [8, 11, 10, 14, 26, 30, 33]. For example, Pielot et al. [33] show that users consider notifications from communication applications (such as messengers and email clients) as important. These notifications are perceived as less annoying and are less likely to be dismissed compared to notifications from other applications. Felt et al. [17] found that the user's perception towards mobile notifications varies strongly. If applications trigger notifications that are not considered relevant, users tend to get annoyed and delete them. Fischer et al. [19] show that users' receptivity is influenced by their general interest in the notification content, entertainment value perceived in it and action required by it, but not the time of delivery. Mehrotra et al. [26] show that notifications containing important or useful content are often accepted, despite the disruption caused by them.

At the same time, past studies have proposed interruptibility management mechanisms for delivering notifications at the inferred opportune moments by using the user's context [23, 21, 20, 18, 32, 31, 29] and the notification content [25]. In [21] Ho and Intille suggested that the transition between two physical activities (such as sitting and walking) can be used as opportune moments for delivering notifications. Pielot et al. [31] proposed a model that can predict whether a user will view a notification within a few minutes with a precision of approximately 81%. Pejovic and Musolesi [29] proposed a mechanism that relies on the contextual information (including activity, location and time of day) to predict opportune moments for delivering notifications. Mehrotra et al. [25] suggested using both the contextual information and the notification content for modeling interruptibility. However, the authors assumed that an application triggers only a single type of notifications. Instead, in this work, for the first time, we design an interruptibility management system that classifies notifications into different classes based on the information they contain and learns the user's preferences for receiving the types of information in different situations.

Most importantly, none of the proposed mechanisms discussed above can be deployed in a real world scenario because their accuracy is not sufficient to ensure that they would never stop or defer important notifications. We have designed a solution

to offer transparency in terms of the interruptibility management mechanism to the users in order to reduce the likelihood of incorrectly scheduling/stopping important notifications and we have evaluated it *in-the-wild*.

## CONCLUSIONS

In this paper we have presented a novel solution for intelligent notification management based on the automatic extraction of rules that reflect user's preferences. The goal of the proposed approach is to make notifications *intelligible* to users. We first evaluate our proposed mechanism with a large-scale dataset of notifications collected during an interruptibility study. Our results show that by using the notification title and the user's location, we can predict if a message will be dismissed by a user with a very high precision.

We have also discussed the design of an open source Android library implementing the interruptibility mechanism and the implementation of the PrefMiner application built on top of it. Through an *in-the-wild* deployment, we have shown that PrefMiner represents a very effective, yet transparent, solution for interruptibility management for mobile devices.

## REFERENCES


1. 2016. Android DDMS. (2016). `http://developer.android.com/tools/debugging/ddms.html`.
2. 2016. CLOC – Count Lines of Code. (2016). `http://cloc.sourceforge.net`.
3. 2016. Facebook `http://www.facebook.com`. (2016).
4. 2016. Google Now. (2016). `http://www.google.com/landing/now/`.
5. 2016. qdapDictionaries for R. (2016). `http://trinker.github.io/qdapDictionaries/`
6. 2016. Snowball Stemming Libraries for R. (2016). `http://snowball.tartarus.org`.
7. 2016. Weka: Data Mining Software in Java. (2016). `http://www.cs.waikato.ac.nz/ml/weka/`
8. Piotr D Adamczyk and Brian P Bailey. 2004. If not now, when?: the effects of interruption at different moments within task execution. In *CHI'04*.
9. Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining association rules between sets of items in large databases. In *SIGMOD'93*.
10. Brian P Bailey and Joseph A Konstan. 2006. On the need for attention-aware systems: Measuring effects of interruption on task performance, error rate, and affective state. *Computers in Human Behavior* 22, 4 (2006), 685–708.
11. Brian P Bailey, Joseph A Konstan, and John V Carlis. 2001. The effects of interruptions on task performance, annoyance, and anxiety in the user interface. In *INTERACT'01*.
12. Michael W Berry and Jacob Kogan. 2010. *Text mining: applications and theory*. John Wiley & Sons.
13. Herbert H Clark. 1996. *Using language*. Cambridge University Press.
14. Edward Cutrell, Mary Czerwinski, and Eric Horvitz. 2001. Notification, disruption, and memory: Effects of messaging interruptions on memory and performance. In *Interact'01*.
15. Nathan Eagle and Alex Sandy Pentland. 2009. Eigenbehaviors: Identifying structure in routine. *Behavioral Ecology and Sociobiology* 63, 7 (2009), 1057–1066.
16. Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *KDD'96*.
17. Adrienne Porter Felt, Serge Egelman, and David Wagner. 2012. I've got 99 problems, but vibration ain't one: a survey of smartphone users' concerns. In *SPSM'12*.
18. Joel E Fischer, Chris Greenhalgh, and Steve Benford. 2011. Investigating episodes of mobile phone activity as indicators of opportune moments to deliver notifications. In *MobileHCI'11*.
19. Joel E Fischer, Nick Yee, Victoria Bellotti, Nathan Good, Steve Benford, and Chris Greenhalgh. 2010. Effects of content and time of delivery on receptivity to mobile interruptions. In *MobileHCI'10*.
20. Sukeshini A Grandhi and Quentin Jones. 2009. Conceptualizing interpersonal interruption management: A theoretical framework and research program. In *HICSS'09*.
21. Joyce Ho and Stephen S. Intille. 2005. Using Context-Aware Computing to Reduce the Perceived Burden of Interruptions from Mobile Devices. In *CHI'05*.
22. Eric Horvitz, Johnson Apacible, and Muru Subramani. 2005. Balancing awareness and interruption: Investigation of notification deferral policies. In *User Modeling*. 433–437.
23. Eric Horvitz, Paul Koch, and Johnson Apacible. 2004. BusyBody: creating and fielding personalized models of the cost of interruption. In *CSCW'04*.
24. Shamsi T Iqbal and Eric Horvitz. 2010. Notifications and awareness: a field study of alert usage and preferences. In *CSCW'10*.
25. Abhinav Mehrotra, Mirco Musolesi, Robert Hendley, and Veljko Pejovic. 2015. Designing Content-driven Intelligent Notification Mechanisms for Mobile Applications. In *UbiComp'15*.
26. Abhinav Mehrotra, Veljko Pejovic, Jo Vermeulen, Robert Hendley, and Mirco Musolesi. 2016. My Phone and Me: Understanding User's Receptivity to Mobile Notifications. In *CHI'16*.

27. Tadashi Okoshi, Julian Ramos, Hiroki Nozaki, Jin Nakazawa, Anind K Dey, and Hideyuki Tokuda. 2015. Reducing users' perceived mental effort due to interruptive notifications in multi-device mobile environments. In *UbiComp'15*.

28. Carlos Ordonez. 2006. Comparing association rules and decision trees for disease prediction. In *HIKM'06*.

29. Veljko Pejovic and Mirco Musolesi. 2014. InterruptMe: designing intelligent prompting mechanisms for pervasive applications. In *UbiComp'14*.

30. Martin Pielot, Karen Church, and Rodrigo de Oliveira. 2014a. An in-situ study of mobile phone notifications. In *MobileHCI'14*.

31. Martin Pielot, Rodrigo de Oliveira, Haewoon Kwak, and Nuria Oliver. 2014b. Didn't you see my message?: predicting attentiveness to mobile instant messages. In *CHI'14*.

32. Stephanie Rosenthal, Anind K Dey, and Manuela Veloso. 2011. Using decision-theoretic experience sampling to build personalized mobile phone interruption models. In *Pervasive Computing*. 170–187.

33. Alireza Sahami Shirazi, Niels Henze, Tilman Dingler, Martin Pielot, Dominik Weber, and Albrecht Schmidt. 2014. Large-scale assessment of mobile notifications. In *CHI'14*.

34. Philip H Swain and Hans Hauska. 1977. The decision tree classifier: Design and potential. *IEEE Transactions on Geoscience Electronics* 15, 3 (1977), 142–147.

35. Yiming Yang and Jan O Pedersen. 1997. A comparative study on feature selection in text categorization. In *ICML'97*.

36. Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao, and Lei Yang. 2010. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In *CODES/ISSS'10*.