INFOCOM 2017 1570298447

Understanding and Managing Notifications

Swadhin Pradhan*, Lili Qiu*, Abhinav Parate†, and Kyu-Han Kim‡
* University of Texas at Austin † Lumme Inc. ‡ Hewlett-Packard Labs

***Abstract*—In today's always-connected world, we receive a large number of notifications on our mobile devices. These notifications cause interruptions, stress, and even impact users' lifestyle. To understand how users respond to notifications, we develop an application that monitors various features (*e.g.*, importance) of the notifications, users' actions, and the level of users' engagement with the notifications. We recruit 30 users to use the application and monitor over 30 days, and subsequently find that 20% to 50% of the notifications generally get ignored by the users. In addition, we also solicit explicit feedback about the importance of notifications from 12 users over 14 days and identify the relation between perceived importance and users' engagement level. Based on this study, we identify the key characteristics of notifications and users' engagement, which is further substantiated by an online survey of 400+ users. In addition, we develop a notification manager that includes a machine learning based prediction model and that shows only the important notifications and delays the unimportant notifications. Our experimental results show that our notification manager automatically assesses the importance of notifications with more than 87% accuracy. We believe this work is a promising step toward intelligent personal assistant that manages notifications.**

I. Introduction

Modern mobile operating systems support *notifications* as a mechanism to present a concise piece of information. Most mobile applications today employ notifications for a variety of reasons through different modalities (Fig. 1). For example, apps like calendar use notifications to alert users about upcoming events, gaming apps use notifications to increase users' engagement with their apps, and email apps let users take quick actions via notifications to reply or archive a new email. Since notifications are designed to be compact and do not require users to launch the full app, notifications have emerged as a popular mechanism to gain user attention. While the notifications are useful, the uncontrolled surge of notifications are known to cause unwanted interruptions [18], distractions, and even stress [30] to the phone users. Studies have shown that users are getting a prohibitively large number of notifications [25], [4], [19], causing delay in on-going tasks [15], and hampering lifestyle.

There have been several prior efforts at addressing some of the issues. They have focused on addressing the question of "*when*" to present a notification such that it causes minimum disruption (*i.e.*, selecting opportune moments to notify users) [15], [11], and results in high user responsiveness (*i.e.*, how fast user react to a notification) [26]. Subsequently, researchers have developed intelligent notification management systems, such as *InterruptMe* [24], which tries to infer opportune moments to interrupt from explicit user inputs and sensor-inferred contexts, and *Atillia* [23], which infers the interruption points automatically without using power-hungry sensors. These studies are very useful, but users still get unwanted notifications though at more appropriate times.

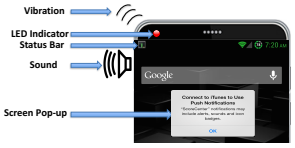


Fig. 1. Different modalities of notifications in today's smartphone.

Our online survey of 400+ users reports 92% users check notifications on smartphones and only 25% of them are completely satisfied with the current notification systems. Furthermore, 66% users are interested in using a new notification manager that automatically filters out unwanted notifications. Clearly there is a need to design a smart notification management system that takes into account the *user-perceived importance* of notifications. Using the perceived importance of notifications as input, such a system can i) suppress or delay a less-important notification to be presented later, ii) re-order the notifications based on the rank of user-perceived importance, and iii) select an appropriate mode of notifying a user (*e.g.*, sound, LED indicator, vibration) based on the notification importance and user context.

The first step towards designing an *importance-aware* notification management system is the ability to learn and predict user-perceived importance of notifications. In this paper, we present a system to predict the likely importance for a new notification generated by an app. This system leverages various contextual information obtained from the phone and the past user interactions with the notifications to predict the importance for an incoming notification. An interesting feature of our system is its ability to passively monitor user-perceived importance of notifications without requiring any explicit user feedback. The key intuition is that a user responds to a useful notification in a meaningful manner (*e.g.*, opening the notification drawer). Based on the passive monitoring, we learn a model that maps contextual features of a notification to its perceived importance.

Our main contributions are summarized as follows:

- We present a user study conducted with 40 users and analyze the data from 30 users having at least two weeks of data to understand the notification usage on phones (Sec. IV). We show that users ignore between 20-50% of the notifications. We also conduct 400+ user online survey through *Amazon Mechanical Turk* to shed more light on the notification usage pattern.

- We present an approach to capture user-perceived importance of notifications (Sec. V). We compare the performance of our approach with the explicit feedback from 12 users. Results illustrate that the explicit user response corresponds well with the predicted implicit importance.
- We predict the importance of an incoming notification using a set of 22 discriminating features. Results show that our model achieves 87% accuracy.

We begin with background describing the existing approaches for managing notifications. Next we describe datasets and gained insights. Then we present the design, evaluation, and implementation of a notification manager prototype.

II. Background

In this section, we outline the existing approaches for notification management and describe their limitations.

OS-level control: Most modern mobile operating systems provide users with the control to block all notifications from chosen apps. Our online survey with 400+ users indicate that 42% of these users filter notifications based on apps. However, this approach provides limited flexibility because there are apps like Facebook that generate a variety of notifications, such as birthday alerts, new message alerts, and so on. Users have different preferences depending on the type of notifications even from the same app. If we build our learning model (Sec. V) only based on *app names*, it yields limited performance: 0.58 precision and 0.59 recall.

Do not disturb mode: This is another control provided by the mobile operating systems, which allows a user to disable all notifications from any apps while the *do not disturb* mode is on. 22% of the surveyed users use this mode as their first choice mechanism to avoid notifications. While this control avoids interruption to users, it does not distinguish between important and unwanted notifications. If *hour of the day* were the only feature in our prediction engine (Sec. V) assuming users put *do not disturb* mode at fixed hours of the day, it would only achieve 0.57 precision and 0.56 recall, which shows the inadequacy of time based mode selection.

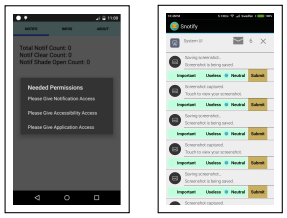
Importance learning approach: Recently, a new set of apps have emerged that incorporate intelligence to determine the importance of a notification for its users. *Inbox by Google* is such an app that manages emails and generates an email notification only when it is considered important. This app predicts email importance based on its content and the past user interaction with the emails. According to our survey, currently around 10% of the users use this type of apps. However, this approach requires developers of each app to develop their own custom experience-sampling approach and a prediction algorithm for their notifications. Our work avoids the issue of app-specific changes to provide a general smart notification management in an easy-to-use and privacy preserving manner.

III. Datasets

We have collected data from users in two phases. In the first phase, we analyze the current situation of notification disruption. To this end, we perform a study by collecting notifications coupled with many smartphone events, interactions associated with notifications, sensor data, and app usage events from 40 users via *Notifbase* app [3] (Fig. 2(a)) (approved by IRB). The users consist of 11 undergraduate students, 23 graduate students, and 6 researchers. Among them, there are 7 females and 33 males. This dataset is called *Dataset - I*. In the second phase, we collect explicit feedback of notification importance by developing an app called *Snotify* [6] (Fig. 2(b)), which collects data from 12 randomly selected users. This app collects usage data along with users' feedback. Furthermore, we have also collected responses from *online survey* [5] involving 400+ users through *Amazon Mechanical Turk* to get an independent subjective understanding of notification usage and management traits.

Dataset I: For this study, we have recruited 40 users by installing *Notifbase* [3] (Fig. 2(a)) in their *Android* smartphones requiring at least version 4.4 (due to the need of use of *NotificationListenerService*). This android app needs *Notification listening*, *Accessibility event listening* and *App usage listening* permissions (Fig. 2(a)). Out of 40 users, we get 30 users with at least 14 days of usage as a few users uninstalled the app or left the study. This dataset includes app-usage, screen on/off, Wi-Fi status, headphone status, coarse location using cellular towers, battery level, notification events (post, clear, action), notification properties (time, title, id, style, modality), notification shade opening or duration (through accessibility service), ringer mode, calendar event, raw data from accelerometer, gyroscope, proximity sensors (only 10 seconds of data recorded after notification posting to save energy) and audio features of decibel and pitch recorded from 10 seconds of data using TarsosDSP library [7] (no raw audio data are recorded for privacy). We have recorded the data in compressed format and sent the data to the cloud server through secure https connections whenever Wi-Fi was available to avoid cellular data usage charge.

Dataset II: To understand how users perceive the importance of different notifications, in the second phase we have collected explicit feedback of importance from users regarding the notifications using *Snotify* app [6] shown in Fig. 2(b)). It also runs on *Android* smartphones requiring at least version 4.4. This app, like *Notifbase*, collect similar set of information. This app requires the participant to give feedback about the perceived importance of the notifications through the interface



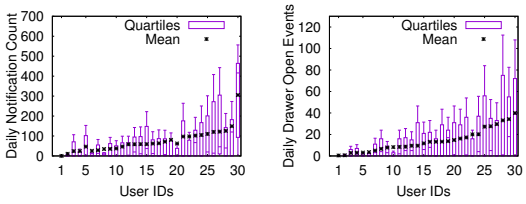
(a) (b)

Fig. 2. (a). *Notifbase* screenshot with its permission dialog. (b). *Snotify* feedback page screenshot.

shown in Fig. 2(b). Users can choose among three levels of feedback: *Neutral*, *Useless*, *Important* (shown in Fig. 2(b)) as feedback for a given notification or no feedback, which will be marked as *None*. Users are regularly reminded to provide feedback.

IV. Understanding Characteristics of Notification Usage and Management

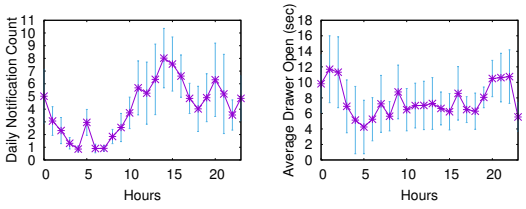
In this section, we present some of the insights gained regarding notification usage based on the analyses done on *Dataset I* from 30 users with more than 2 weeks of data. In particular, we examine the user behavior while engaging with notifications. And then we ask: are app developers good at judging the importance of notifications for its users?



(a) Notification Received (b) Notification Checking

Fig. 3. Daily notification events across 30 users in **Dataset I**. Unless otherwise specified, all the box plots shown in this paper plot the mean at the center and uses the length of the bar to reflect the $1.5 \times InterQuartileRange$, whereas error bars in line or bar plots reflect standard deviation.

How frequently users receive and check their notifications? First we look at the number of notifications received by users in our study. Fig. 3(a) shows the distribution of daily number of notifications received for each user in our study. We see that an average user receives at least 60 notifications per day, similar to the median value of 55 notifications as reported by users in *Online survey*. Some users get up to 600 notifications a day. Fig. 4 shows that a large fraction of these notifications are received during the waking hours of the day between 8 AM - 10 PM. If not managed properly, these notifications have the potential to disrupt an average user at least 4 times per hour of the productive part of their day. Furthermore, we found that only 5% of all the apps are responsible for 70% of the notifications. These apps include apps like *WhatsApp*, *WeChat*, *Instagram*, *Gmail*, *Facebook Messenger* etc.



(a) Hourly Distribution of Average Notifications Received (b) Hourly Distribution of Notification Checking Events

Fig. 4. Majority of the notifications are received during the waking hours between 8 a.m. - 10 p.m. While users receive more notifications during the early afternoon hours, they check notifications more afternoon onward.

Next, we examine how frequently users pay attention to notifications in their daily lives. To examine this, we monitor how frequently users check for the notifications by opening the notification drawer on their phones. Fig. 3(b) shows that an average user opens notification drawer at least 20 times a day, and some users can check notifications as high as 260 times a day which is still much lower than the number of notifications received. We also found that an average user spends 15 *minutes* per day checking notifications. From Fig. 4, we see that while users receive more notifications during early afternoon hours, they check their notifications more frequently late afternoon onward, as shown in Fig. 4(b). In other words, *the user receptivity towards incoming notifications varies during the course of the day*. This is further substantiated when we observe the duration where users put their phones into less-disturbing modes such as *Silent* or *Vibration*. Fig. 5 shows that users spend the major part of their day in less-disturbing modes indicating their willingness to avoid or tolerate delays in checking notifications.

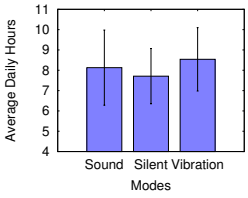


Fig. 5. Average daily hours in a specific mode (Silent mode/ Vibration mode/Audible mode). Users put phones in silent mode for nearly 8 hours.

How do users engage with notifications? There are several ways a user can engage with a notification. Understanding this user engagement behavior is critical for designing an effective notification manager. We classify the user engagement behavior into the following classes:

- *Read:* A user spends some time reading the notification but does not take any other actions. A user may read it either in a notification drawer along with other notifications, or on a lock screen if the notification is visible. We mark notifications with such user engagements as *Seen*.
- *Read and dismiss:* In this case, a user spends some time reading a notification and dismisses that notification individually. To collect this information, we look for a *Dismiss* event (from Accessibility service) coupled with a *Notification Clear* event (through Notification Listener Service) whenever a notification drawer is open. We tag this engagement level as *Dismissed* only if the notification drawer had been open for at least 5 seconds.
- *Take actions:* Many notifications provide users with an embedded set of buttons to take quick actions. For example, Email notifications come with actions, such as "Archive" or "Done". We use Accessibility Service to monitor if user executed any *action* on a particular notification whenever a *notification drawer* is open. We mark such instances as *Actions*.
- *Launch an app:* This is the highest level of engagement, where a user launches an app corresponding to the notification. To record this user engagement, we monitor notifica-

tion clear events triggered by a user click on a notification. If we notice launching of an app, corresponding to the clicked-upon notification, within 2 seconds of the click event, we mark this engagement as *Launched*.

- *Ignore:* A user may ignore the notification, *i.e.*, none of the previous engagements have occurred. In such a scenario, an application may remove the notification after a timeout, or it may update the notification with new content later. We mark a notification instance as *Ignored* if we do not observe any of the above engagement, or when we observe a notification clear event triggered without any user intervention.

Fig. 6 plots the distribution of the various engagement classes for each user in our dataset. It shows that i) 20%-50% notifications are ignored by users, ii) 30% notifications lead to the highest level of engagement (*i.e.*, launch an app), and iii) users pay some attention to around 60% of notifications.

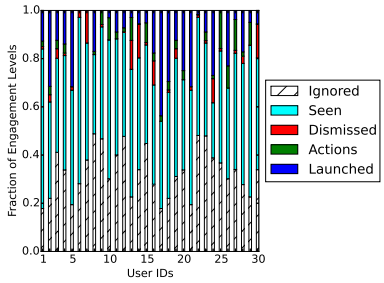
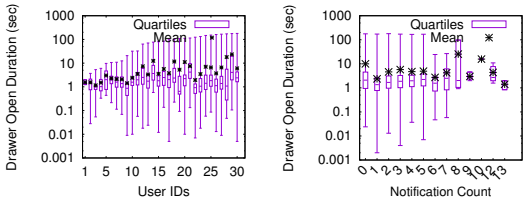


Fig. 6. Distribution of user engagement classes for 30 users. Users ignore between 20 to 50% of all notifications.

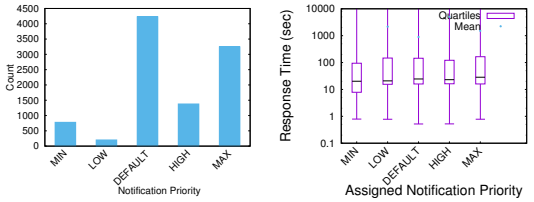
How much is the attention span for reading notifications? Now we study the user attention span i.e. the contiguous chunk of time a user devotes to reading notifications . Fig. 7 reveals that users generally spend around 1 to 20 seconds within a notification drawer to browse through different information posted. Moreover, this duration does not vary much with the number of notifications in a drawer (Fig. 7(b)). This signifies that the limited user attention should be used judiciously to present important notifications first.



(a) Drawer duration vs User (b) Drawer duration vs No. of notifications in a drawer

Fig. 7. Distribution of the duration when a notification drawer is open for reading notifications. An average user spends 10 seconds reading notifications. This duration does not vary with increase in number of notifications.

Are app developers good at evaluating importance? Android operating system provides app developers to assign a priority level for a notification. The developers can assign 5 levels of priority ranging from MIN to MAX depending on how important they judge a notification is for the end user. Not surprisingly, most of the developers choose either $DEFAULT$ value or the higher priority values to attract users' attention, as shown in Fig. 8(a). Moreover, Fig. 8(b) shows that the user response time does not vary much with the notification priority. Therefore, it is not very helpful to always use highest priorities to attract users' attention.



(a) Distribution of notification priority assigned by app developers. Most notifications are assigned the default priority.

(b) User response time for different priority notifications which remains almost constant.

Fig. 8. Default priority of notifications and user response.

A. Summary

Our detailed analyses reveal the following observations: (i) Users receive a large number of notifications: an average of 60 per day. (ii) Users take actions to prevent disruption from many notifications by putting phones in silent mode. (iii) Users engage with notifications in various ways and ignore many notifications (20 - 50%). This motivates the need of automatically filtering unimportant notifications. (iv) Users have a limited attention budget ($<$ 10 seconds). Motivated by this insight, we show notification digest in a ranked order (Sec. VII). (v) Apps tend to assign overly high priorities to notifications, but these priorities do not significantly affect users' response time. All these observations lead to the conclusion that there is a need for automatically ranking notifications based on its importance so that users are disrupted less frequently and the limited user attention can be used judiciously.

V. Notification Importance Prediction

In this section, we describe our approach based on *supervised* classification to predict the importance of a notification. To train such a prediction model, we need feedback from the user providing ground-truth about the user-perceived importance for each notification. However, soliciting explicit feedback for each notification is cumbersome for the user. Instead, our notification manager leverages a combination of user-control method and a passive monitoring of user engagement with any notification to implicitly infer its importance. In the following, we first describe our approach to monitor notification importance. Next, we present the contextual features we use in our prediction model. Finally, we describe our machine-learning based model to predict the user engagement.

A. Notification Importance Monitor

In this section, we describe our approach to capture user-perceived importance using the empirical data collected from

TABLE I
COMPARING FEEDBACK AND ENGAGEMENT LEVELS

	Dismiss	Action	Launch	Seen (Response Time $\leq$10s)	Seen (Response Time $>$10s)	No Action
Useless	**66**	**56**	**76**	**78**	*379*	*785*
Important	*161*	*117*	*253*	*521*	**192**	**218**

12 users in Dataset II where users contributed explicit feedback for 4085 notifications over a period of at least 2 weeks.

User Control: In our dataset, we observed that the notifications generated by a set of apps, like calendar (*android calendar* etc.), weather (*accuweather* etc.), specific to a user, were always labeled as *important*. We had 279 such instances in our data. Similarly, we observed another set of apps, like messenger (*facebook messenger*, *whatsapp* etc.) or mail based applications (*gmail*, *yahoo mail* etc.), specific to each user, whose notifications were *never* perceived to be important. There were 285 such instances in our dataset. Based on these observations, we propose presenting a user with an interface where they can identify the set of apps whose notifications are always (or never) important for them. In fact, our survey among the 12 participants confirms this observation. Using this approach, we know the consistent importance of notifications for the selected set of apps.

Rule-based Implicit Feedback Monitor: For the set of apps that are not selected above, we use a rule-based approach to implicitly infer notification importance. We learn these rules using the empirical data from our dataset. To learn these rules, we first remove the notifications for apps from our dataset that are always or never important. Next, we remove another 1180 instances where the users were *neutral* about the importance. For the remaining 2341 instances, we extract the user engagement with the notifications (ignored, read, dismissed, acted upon, app launch), the time spent reading the notification, and the time taken to respond to the notification as the features in our classifier. Using these features, we learn a rule-based classifier that classifies a notification as *important* if one of the following conditions is met: notification is dismissed after reading for at least 5 seconds, any action is taken on the notification, a corresponding app is launched soon after the notification is received, or the notification is read within 10 seconds of its arrival. The rest is considered as *ignored*. Our rule-based classifier results in accurately classifying 86.29%.

B. Feature Extraction

Next we describe a set of 22 features (refer Table II) we use in our model to predict importance of notifications. These features are classified as follows:

Temporal Features: A typical user behavior in terms of notification engagement varies throughout the day, as shown in Section IV. Based on the time when a notification is posted, we derive features, such as hour of the day, time segment of the day (4 hours segment in a day), and weekends vs. weekdays. For example, a work email notification during the working hours is considered important but the same notification during off-work hours may not be perceived as important.

User device activity based features: A user's recent activities on the phone can be a good indicator of whether the user is going to pay attention to a notification. In particular, we use the following features: (i) time elapsed since the last notification was posted by any app or the same app, (ii) time elapsed since the last time user interacted with some notification or the same kind of notification, (iii) time elapsed since the last time user used the mobile device, (iv) operating mode of the phone: silent, vibration, or audible mode, (v) time elapsed since the last time user used the app that generates the current notification.

Location based feature: There are certain type of notifications that are found to be useful in specific locations. For example, a notification reminding a person to play a game is unlikely to be considered important at a workplace. Since the exact user location is privacy sensitive, we use the user's coarse location based on *cell-tower triangulation* and cluster the geographic locations into "places" of radius 200 meters, and label these clusters as locations 1, 2, 3, etc.

Sensor data based features: Mobile devices come with several sensors, such as microphone, accelerometer, and gyroscope. Using microphone data, we evaluate if a user is in a noisy environment (more than 55 decibels) or not. In noisy scenarios, a user is expected to be less responsive to the notifications. Similarly, data from accelerometer and gyroscope gives user context, such as if a user is walking or stationary. We calculate the number of peaks in the smoothed accelerometer data to detect *walking* (more than 2).

Notification based features: This set of features derived from the notification content distinguish between the types or the *classes* of notifications generated by an individual app. Unlike [20], we do not record plain text of notification content to protect user's privacy. Moreover, the participants in our *Field study* and *Online survey* had expressed some kind of reservation against sharing content of the notifications (more than 60%). Thus, we only record *notification identifier* (a numeric identifier used by the app), an *icon identifier* (numeric identifier of the icon displayed), title of the notification, and one-way hash of words in the summary text of notification. Many mobile applications use a unique notification identifier or a unique icon to display the notifications for each type. For example, Facebook mobile app uses a unique identifier to display birthday reminders but uses another unique identifier for a notification of a different type, such as wall post alert.

Feature Ranking: To understand the value of these features, we rank each feature based on the information gained by adding it for predicting the notification importance. We use the *InfoGainAttributeEval* method from WEKA to derive the information gain (IG) each of the attributes brings to the overall binary classification of a notification as important or not. Table II shows the average information gains of the features. Our results show that the temporal features like hour of the day or temporally local event based feature like last app use are the most important features. Furthermore, notification property driven features, location and calendar events are also relatively important features in terms of information gain results. However, activity level or sensor based features are not that important, which suggests that we can build a good

TABLE II
FEATURES USED

Feature	ID	Average IG
Application name of notification	**1**	**0.326**
Time elapsed since last notification clear time	**2**	**0.226**
Time elapsed since last notification post time	**3**	**0.204**
Time elapsed since last any app use time	**4**	**0.201**
Time elapsed since last any notification post time	**5**	**0.165**
Time elapsed since last app use time	**5**	**0.159**
Hour of the day	**6**	**0.142**
Notification title	**7**	**0.122**
Ringer mode of phone	**8**	**0.113**
Time elapsed since last this notification clear time	**9**	**0.099**
Time elapsed since last screen on time	**10**	**0.095**
Segment of the day	**11**	**0.084**
Weekend status	**12**	**0.081**
Location cluster	**13**	**0.063**
Notification Icon	**14**	**0.060**
Calendar event status	**15**	**0.049**
Last notification engagement level	**16**	**0.048**
Notification clearable status	**17**	**0.031**
Proximity status	**18**	**0.024**
Audio Level	**19**	**0.019**
Activity Level	**20**	**0.009**
Notification assigned priority	**21**	**0.006**
Notification word count	**22**	**0.001**

enough intelligent notification manager without continuously recording sensor data.

C. Machine Learning based Prediction

We use the following machine learning algorithms in our notification importance prediction: (i) $C4.5$ based Decision Tree model, (ii) random forest model, (iii) linear regression model, (iv) support vector machine (SVM), (v) neural networks, (vi) online learning with different loss functions in *Weka* [9], Matlab (for neural network), and vowpal-wabbit [8] to predict if the notification is important or not. Recall that a notification is deemed important if a user interacts with the notification (takes an action, launches an app, reads it for at least 5 seconds, or responds to it within 10 seconds of its arrival. We train the learning models using 22 extracted features. Decision tree uses the information gain to select the feature at each branch for classification. Random forest tries to average multiple decision trees to reduce variance. Linear regression tries to find the weight of each feature by solving $A_{train}x = b_{train}$, where A_{train} and b_{train} are the feature values and ground truth (*e.g.*, 1 for interacted and 0 for ignored) in the training data, respectively. After we estimate x using the training data, which is the feature weights, we apply $A_{test}x$ to estimate b_{test} in the testing data. SVM is another popular algorithm, which tries to divide different categories of samples by maximizing their gaps. Neural network is also a well-known machine learning algorithm, which can support continuous non-linear functions. We use a large portion of training data to train multiple neural networks, each with different parameters, and then use the remaining portion of the training data to compute the estimation error and select the neural network parameters that yields lowest error. This is possible since we know the ground truth of all training data. The neural network is trained using Levenberg-Marquardt algorithm [22]. We use 'mapminmax' to normalize the inputs, use 'tansig' transfer function for hidden layers, and use 'purelin' as the output transfer function. Furthermore, we also implement different online learning models [8], which basically updates weights of the model with every new example (*i.e.*, $w_{new} \leftarrow w_{old} + \eta \times (y - \hat{y})$ (where $\hat{y}$ is the predicted class, y is the actual class, η is the learning rate, and $(y - \hat{y})$ is a simple loss function). We have experimented with stochastic gradient descent method ($w_{new} \leftarrow w_{old} - \eta \Delta Q_i(w)$), hinge loss function (max(0, $(1 - t) \times \hat{y}$)), and default squared loss function ($(y - \hat{y})^2$) with different norms (L1 and L2).

VI. EVALUATION

In this section, we use notification importance prediction model to predict two classes *interacted* and *ignored*. We use the following metrics to evaluate our model.

Accuracy: This metric measures the fraction of notifications of which engagement levels are correctly predicted (number of correctly predicted notifications as *interacted* (important) or *ignored* (unimportant) / total number of notifications).

Precision: It calculates the fraction of notifications which are correctly predicted as *interacted* (*i.e.*, number of notifications that are correctly predicted as *interacted* / number of notifications that are predicted as *interacted*). This indirectly measures the disruption to the users because high precision means that user will get less *ignored* notifications.

Recall: It measures the fraction of *interacted* notifications that are actually predicted correctly (*i.e.*, the number of notifications that are correctly predicted as *interacted* divided by the number of notifications that are actually *interacted*). High recall means that users will not miss out on any *interacted* (important) notifications.

F-Score: This measure is a combination of *Precision* and *Recall*, which is calculated as $\frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$. For all four metrics, larger values indicate higher accuracy.

A. Evaluating the Personalized Predictors

First, we evaluate the data-driven prediction models by testing with the *k-fold cross validation* approach with $k = 10$. That is, we randomly partition our notification dataset of each user (30 users have contributed total $78,485$ samples) (from **Dataset I**) into k equal sized sub-samples and then out of the k sub-samples, a single sub-sample is retained as the validation data for testing the model, and the remaining $k-1$ sub-samples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k sub-samples used exactly once as the validation data. The k results from the folds can then be averaged to produce a single estimation. So, this is a personalized k-fold prediction performed on each user. We use four machine learning techniques from *Weka* library, namely *Random Forest* with 100 trees, *Decision Tree*, *Support Vector Machine (SVM)*, and *Linear Regression* models for predicting notification engagement classes, *i.e.interacted* and *ignored*. As shown in the Fig. 9, the four techniques trained with our 22 features perform similarly in terms of different metrics. Overall average accuracy of prediction is more than 87%. The result also shows that precision and recall

are more than 87%, which signifies that we will be able to suppress unwanted notifications by not missing the probable candidates of important notifications. *Random Forest* performs best in terms of all four metrics and also shows comparably less variation across users. On the other hand, decision tree model performs best in terms of time without compromising too much on accuracy. Therefore it has been selected for our smartphone prototype implementation. In comparison, SVM is the slowest.

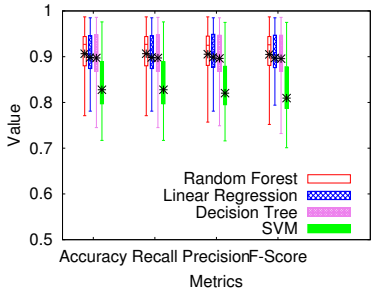


Fig. 9. Box-plot of Accuracy, Precision, Recall, and F-score of engagement level prediction techniques of 30 users.

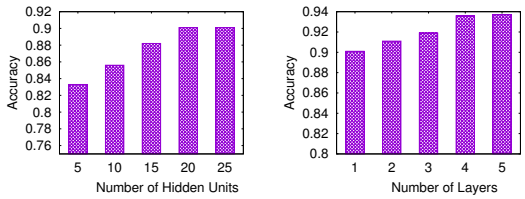
B. Generic vs Personalized vs Clustered Predictors

Next we compare the performance of the engagement prediction model trained not only on a user's personal data but also with a generic prediction model trained on multiple users' data and with a prediction model trained on clustered user data. As we are not collecting any personal information from the users due to privacy concern, we cluster the users using K-means clustering algorithm based on the following feature set (normalized): *number of applications used*, *number of unique locations visited*, *number of notifications received*, and *number of each engagement levels*. We build both personalized predictors, generic predictors, and clustered predictors for predicting notification importance by using four aforementioned machine learning algorithms. For personalized prediction, we use 10-fold cross validation. For generalized prediction, we partition the data into two sets: training (containing 45,085 samples from randomly selected 15 users) and testing (containing 33,400 samples from rest of the 15 users). For clustered based prediction, we first cluster 30 users and employ 10-fold cross-validation approach within each cluster (and report the average result). As Table VI-B illustrates, expected generalized model performs the worst across different measures in different machine learning techniques. However, surprisingly, even this naive cluster-based model performs better than the personalized model. It is mainly because the cluster-based model uses more training data. Unlike the generic model, the training data in the cluster-based model are more similar to the user's personal data. This result is encouraging and indicates that clustering provides a good sweet spot to benefit from both the increase in the training data and the similarity between the training data and testing data. Its benefit is even larger when we consider users with little or no training data.

Furthermore, we build a neural network model using different numbers of hidden units and different numbers of hidden layers to predict the importance of notifications to test in cluster-based situation. Fig. 10(a) reflects that the result does not improve much after 15 hidden units for a single layer, but the increase of hidden units do help in improving accuracy. If we increase the number of hidden layers (each having 20 hidden units), we also see an increase in accuracy. Fig. 10(b) illustrates that we get around 92% accuracy if we use 5 hidden layers, which is a little better compared to decision tree based or random forest based models.

TABLE III
ENGAGEMENT LEVEL PREDICTION USING GENERALIZED, PERSONALIZED, AND CLUSTERED TECHNIQUES

ML Technique	Accuracy	Recall	Precision	F-Score
Generalized Random Forest	0.79	0.79	0.79	0.79
Clustered Random Forest	**0.95**	0.95	0.95	0.95
Personalized Random Forest	**0.91**	0.91	0.90	0.89
Generalized Linear Regression	0.78	0.77	0.76	0.75
Clustered Linear Regression	**0.92**	0.92	0.92	0.92
Personalized Linear Regression	**0.88**	0.88	0.87	0.88
Generalized Decision Tree	0.80	0.80	0.80	0.80
Clustered Decision Tree	**0.93**	0.93	0.91	0.91
Personalized Decision Tree	**0.88**	0.88	0.88	0.89
Generalized SVM	0.72	0.73	0.72	0.73
Clustered SVM	**0.86**	0.86	0.85	0.84
Personalized SVM	**0.80**	0.80	0.80	0.80



(a) Multiple Units (b) Multiple Layers

Fig. 10. (a) Average prediction accuracy across users using Neural Network with different number of hidden units in a single hidden layer. (b) Average prediction accuracy across users using Neural Network with different number of hidden layers (having 20 hidden units per layer).

C. Online Learning based Prediction

Fig. 11 shows the prediction accuracy of online learning based models with different training batch sizes. In real world cases, we may use online learning based methods for any of the above discussed strategies (generalized, clustered, or personalized) as new samples are coming in continuously. We use vowpal-wabbit [8] tool to implement and test the feasibility of an online learning system. We have tried different loss functions (like stochastic gradient descent, hinge etc.) with different norms (L1 and L2) in our implementation. However, on an average (with different batch sizes), we get around 60% to 75% prediction accuracy using different online learning methods on our dataset. As we would expect, the accuracy is around $10\% - 20\%$ lower than the offline clustered or personalized learning models. Such accuracy can still help to build a notification manager which can at least filter out a considerable amount of unwanted notifications. The trend in Fig. 11 shows that the model generally stabilizes when

the batch size is around 5000. Furthermore, SGD based loss function performs better than other loss functions. It is due to small batch sizes forces over-fitting, whereas larger batch sizes impose over-generalization.

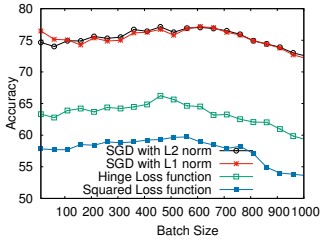
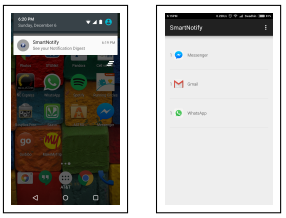


Fig. 11. Average progressive validation loss across users on combined data with different iterations using different Online learning techniques with different batch sizes.

VII. *SmartNotify*: A Notification Manager

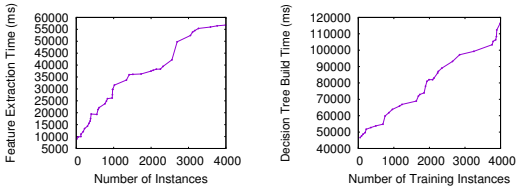
Here, we present a simple notification manager (Fig. 12) implementation and its preliminary evaluation. It pushes important notifications based on engagement prediction model. This scheduler assesses the importance of notification through the predicted engagement level of the notifications. The main components are *Context Monitor*, which collects the contextual data like location, phone events etc., *Usage Monitor*, which logs the notification interaction events, *Importance Predictor*, which predicts the notification importance using Weka [10] by creating a C4.5 decision tree, and *Notification Scheduler*, which shows the notifications in a ranked order based on their engagement level in opportune moments. For simplicity, this correct opportune moment is implemented by looking at the application switch after regular intervals [24].

There are two ways to run the notification manager, which offers a trade-off between privacy and efficiency. One way is to run everything on the phone including building a classifier using the training data (stand-alone). Another way is to let the phone extract features, send them to a server, get the classifier back from the server [1], and apply it to the incoming notifications (client-server). We have implemented both on Samsung Galaxy S3 having 1 GB RAM, Quad-core 1.4 GHz Cortex-A9 CPU (Android 4.4). We have calculated the running time with internal code time stamps and measured the power consumption through Monsoon power monitor [2]. As shown in Fig. 13, for both the *stand-alone* and *client-server* versions, the feature extraction from 4000 instances can be accomplished within 57 seconds and consumes only 0.07% of battery. In practice, the feature extraction is not done in a single batch instead, but inferred passively upon the new notification arrival taking only 14.25 milliseconds per notification on an average. However, for the stand-alone version, the decision tree building can take up to 2 minutes as shown in Fig. 13(b), consuming around 23 Joule of energy per run. We can save this delay if we can opportunistically send data to server. This also saves power because it will consume only around 18 Joule of energy through a client-server data manager.



(a) (b)

Fig. 12. (a). *SmartNotify* Notification Digest Indicator. (b). Front-page of *SmartNotify* app showing ranked notifications.



(a) (b)

Fig. 13. (a). Run-time of Feature Extraction in *SmartNotify* (for both *stand-alone* and *client-server*). (b). Run-time of decision tree formation in *SmartNotify* app (for stand-alone).

Next, we also look at how much notifications we can suppress if we use *SmartNotify*. On an average, we can suppress around 45% of the generated notifications. If we tweak our system to suppress more notifications, we might fail to show important notifications instantly or vice versa, as illustrated in Fig. 14. So, learning the sweet spot of suppression vs. accuracy is important but left as a future work. However, to guard against *missing out anxiety*, we have provided an option in the app so that users can check all the notifications.

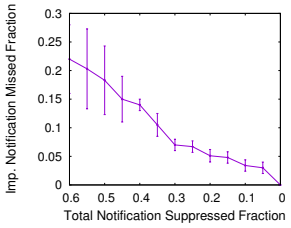


Fig. 14. Trade-off between suppressing notifications and missing out on important notification instantly.

VIII. Related Work

Horvitz et al. inferred interruptibility accurately on desktops by using context information, such as application usage, visual and acoustical cues [15]. Iqbal et al. built OASIS [14], [16], which defers desktop notifications until it detects suitable interruption moments in real-time. More recently, interruptibility research has also focused on mobile devices, with detailed study of notification usage on smartphones [26], [28], [29]. Mehrotra et. al. [21] improved this predictive interruption method on smartphones using more contextual information. Pejovic et al. [24] identifies appropriate moments to interrupt using location, time of day, emotion, and response time. However, this system requires information provided

manually. Attilia [23] automatically mines important usage information to predict opportune moment of interruption. All these interruptibility-based notification management systems in these works focused primarily on understanding the interruption or the opportune time to interrupt. However, a recent work [19], attempts to create association rules based on the content of the notifications to reduce the notification load. But, this system, unlike us, relies on explicit user input combined with privacy-sensitive notification content.

Although interruptibility has been extensively studied, the effect of user's response to the notifications is under-explored. However, previous work [25] has shown that the perception towards mobile notifications varies strongly. If apps which are not perceived as useful keep sending notifications, users become annoyed and consider deleting those apps [12]. According to a field study by Fischer et al. with 11 coworkers[13], the user's responsiveness is determined by message content (*e.g.*, how interesting, entertaining, relevant, and actionable a message is). Unlike one may expect, the time of delivery does not affect responsiveness. Sahami et al. [27] conduct a large-scale study with more than $40,000$ users, and found that notifications from communication applications are considered the most important which is also found true in a recent application usage study [17]. Furthermore, the closest to our work is a recent work by Merhotra et. al. [20], which investigated the connection between notification type and user acceptance. However, they have only gathered information through artificial notification replays in controlled settings while logging only coarse level of user engagement (*e.g.*opening and dismissing an notification). They have not also built a smarter notification manager based on the insights.

IX. Conclusion and Future Works

In this paper, we observe that users get too many notifications on a daily basis, as app developers tend to assign the highest priorities to most of the notifications. However, we observe that we can suppress a large number of unwanted notifications, if we can predict how users might engage with the notifications. We develop a simple notification manager that uses this machine learning based prediction engine to infer the importance of the notifications. Our evaluation shows it can achieve on an average 87% accuracy. Moreover, the core prediction engine can also be used by other apps as a service to determine the acceptability of a notification.

This notification importance prediction model can be used in deciding on the display order of notification, modality of notification (*i.e.*, sound, vibration, or LED indication), or selecting the device (*e.g.*, smartwatch or smartphone) to display upon. We are also interested in enhancing the accuracy of our prediction by using more fine-grained user activities and notification features. Meanwhile, we also want to minimize the processing of personal data while achieving energy efficiency.

References

[1] Exporting randomforest models to java source code. http://pielot.org/tag/export/.

[2] Monsoon power monitor. https://www.msoon.com/.

[3] Notifbase app. https://www.cs.utexas.edu/~swadhin/Notification/Notifbase.apk.

[4] Notifications are the next platform. http://techcrunch.com/2015/04/21/notifications-are-the-next-platform/.

[5] Online survey on notifications. https://www.surveymonkey.com/r/Q3PMNSB.

[6] Snotify app. https://www.cs.utexas.edu/~swadhin/Notification/Snotify.apk.

[7] Tarsosdsp codebase. https://github.com/JorenSix/TarsosDSP.

[8] Vowpal-wabbit : Fast online learning tool. http://hunch.net/~vw/.

[9] Weka 3: Data mining software in java. http://www.cs.waikato.ac.nz/ml/weka/.

[10] Weka for android. https://github.com/rjmarsan/Weka-for-Android.

[11] B. P. Bailey, J. A. Konstan, and J. V. Carlis. The effects of interruptions on task performance, annoyance, and anxiety in the user interface. In *Proceedings of INTERACT*, volume 1, 2001.

[12] A. P. Felt, S. Egelman, and D. Wagner. I've got 99 problems, but vibration ain't one: A survey of smartphone users' concerns. In *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, New York, NY, USA, 2012. ACM.

[13] J. E. Fischer, N. Yee, V. Bellotti, N. Good, S. Benford, and C. Greenhalgh. Effects of content and time of delivery on receptivity to mobile interruptions. In *ACM MobileHCI*, New York, NY, USA, 2010.

[14] S. T. Iqbal and B. P. Bailey. Oasis: A framework for linking notification delivery to the perceptual structure of goal-directed tasks. *ACM Trans. Comput.-Hum. Interact.*, 2010.

[15] S. T. Iqbal and E. Horvitz. Notifications and awareness: A field study of alert usage and preferences. In *ACM CSCW*, 2010.

[16] S. T. Iqbal and E. Horvitz. Notifications and awareness: A field study of alert usage and preferences. In *ACM CSCW*, New York, NY, USA, 2010.

[17] J. Lee, K. Lee, E. Jeong, J. Jo, and N. B. Shroff. Context-aware application scheduling in mobile systems: What will users do and not do next? In *ACM UbiComp*, 2016.

[18] H. Lopez-Tovar, A. Charalambous, and J. Dowell. Managing smartphone interruptions through adaptive modes and modulation of notifications. In *ACM IUI*, New York, NY, USA, 2015.

[19] A. Mehrotra, R. Hendley, and M. Musolesi. Prefminer: Mining user's preferences for intelligent mobile notification management. In *ACM UbiComp*, 2016.

[20] A. Mehrotra, M. Musolesi, R. Hendley, and V. Pejovic. Designing content-driven intelligent notification mechanisms for mobile applications. ACM UbiComp, 2015.

[21] A. Mehrotra, J. Vermeulen, V. Pejovic, and M. Musolesi. Ask, but don't interrupt: The case for interruptibility-aware mobile experience sampling. ACM UbiComp/ISWC Adjunct, 2015.

[22] J. J. Moré. The levenberg-marquardt algorithm: Implementation and theory. In *Numerical Analysis*, Lecture Notes in Mathematics. Springer Berlin Heidelberg, 1978.

[23] T. Okoshi, J. Ramos, H. Nozaki, J. Nakazawa, A. Dey K., and H. Tokuda. Attelia: Reducing user's cognitive load due to interruptive notifications on smart phones. In *ACM PerCom*, St. Louis, Missouri, USA, 2015.

[24] V. Pejovic and M. Musolesi. InterruptMe: Designing Intelligent Prompting Mechanisms for Pervasive Applications. In *ACM UbiComp*, Seattle, WA, USA, 2014.

[25] M. Pielot, K. Church, and R. de Oliveira. An in-situ study of mobile phone notifications. In *ACM MobileHCI*, New York, NY, USA, 2014.

[26] A. Sahami Shirazi, N. Henze, T. Dingler, M. Pielot, D. Weber, and A. Schmidt. Large-scale assessment of mobile notifications. In *ACM CHI*, New York, NY, USA, 2014.

[27] A. Sahami Shirazi, N. Henze, T. Dingler, M. Pielot, D. Weber, and A. Schmidt. Large-scale assessment of mobile notifications. In *ACM CHI*, 2014.

[28] D. Weber, A. S. Shirazi, and N. Henze. Towards smart notifications using research in the large. In *ACM MobileHCI*, 2015.

[29] D. Weber, A. Voit, P. Kratzer, and N. Henze. In-situ investigation of notifications in multi-device environments. In *ACM UbiComp*, 2016.

[30] S. Yoon, S.-s. Lee, J.-m. Lee, and K. Lee. Understanding notification stress of smartphone messenger app. In *ACM CHI*, New York, NY, USA, 2014.