POLITECNICO DI TORINO
Repository ISTITUZIONALE

A Context and User Aware Smart Notification System

Original
A Context and User Aware Smart Notification System / Corno, Fulvio; DE RUSSIS, Luigi; Montanaro, Teodoro. - STAMPA. - (2015), pp. 645-651. (Intervento presentato al convegno IEEE 2nd World Forum on Internet of Things (WF-IoT) tenutosi a Milan, Italy nel 14-16 December 2015) [10.1109/WF-IoT.2015.7389130].

Availability:
This version is available at: 11583/2627751 since: 2016-01-27T20:30:34Z

Publisher:
IEEE

Published
DOI:10.1109/WF-IoT.2015.7389130

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

12 October 2024

Author's preprint. Paper accepted for presentation at
The 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)
14-16 December 2015, Milan, Italy

A Context and User Aware Smart Notification System

Fulvio Corno*, Luigi De Russis*, and Teodoro Montanaro*
*Politecnico di Torino Corso Duca degli Abruzzi, 24 Torino, Italy 10129
{fulvio.corno, luigi.derussis, teodoro.montanaro}@polito.it

***Abstract*—Nowadays, notifications are increasingly gaining momentum in our society. New smart devices and appliances are developed everyday with the ability to generate, send and show messages about their status, acquired data and/or information received from other devices and users. Consequently, the number of notifications received by a user is growing and the tolerance to them could decrease in a short time. This paper presents a smart notification system that uses machine learning algorithms to adequately manage incoming notifications. According to context awareness and user habits, the system decides: *a*) who should receive an incoming notification; *b*) what is the best moment to show the notification to the chosen user(s); *c*) on which device(s) the chosen user(s) should receive the notification; *d*) which is the best way to notify the incoming notification. After the design of a general architecture, as a first step in building such a system, three different machine learning algorithms were compared in the task of establishing the best device on which the incoming notification should be delivered. The algorithms were applied to a dataset derived from real data provided by the MIT Media Laboratory Reality Mining project, enriched with additional synthetic information.**

***Keywords*—Notifications; Machine Learning; Internet of Things;**

I. Introduction

During the last five years, we were faced by a rapid spread of smart "things," physical objects that are always connected to the Internet and are able to accomplish new tasks, like tracking fitness-related metrics (e.g., distance walked or run, calories used up, heartbeat and quality of sleep). New devices and appliances are developed and released every day: wearable fitness bracelets, smart washing machines, smart ovens, and smart thermostats are only a few of the existing Internet of Things (IoT) devices.

One of the features provided by all smart devices is the possibility to generate notifications and, in some cases, to receive and show notifications sent by other services/devices. For instance, if the smart washing machine has just finished the washing cycle and a smart TV is registered as a receiver of messages generated by it, a notification will be shown on both the smart TV and the washing machine itself. Moreover, online social services like Facebook can generate a great number of notifications on all connected devices, e.g., they can notify the user about an incoming message or a friend's new post.

Thus, the number of incoming notifications is growing and the benefit of displaying the same notification on all available devices could put user patience to a hard test. To better understand the problem, we can look at a simple scenario in which a user is in her house and owns a smartphone, a smartwatch, a tablet, a smart thermostat and some other IoT devices. Let's assume that the thermostat app is installed on all the 3 available devices and that in a specific moment of the day the thermostat recognizes a low battery status. Consequently, it sends a message to inform the user. As a result, the user receives 3 different notifications, one on each connected device, for a single message: this replication can be reasonably boring or maybe distracting. Moreover, this trouble becomes more problematic in more complex situations in which, instead of having one single user and one single IoT device, we have multiple users and several IoT devices. The problem of overwhelming notifications is supported by the study conducted by Church et al. [1] on the reasons and perceptions of WhatsApp, a popular mobile messaging application, in which some interviewees declared they were annoyed with the amount of notifications received by mobile messaging applications in general. Moreover, the large-scale assessment of mobile notifications made by Sahami et al. [2] demonstrates that, even though not all the notifications are equally valued by users, they are becoming pervasive and sometimes they reduce users overall performance distracting them from other tasks. Furthermore, they observed that the users' reaction to notifications changes according to what she was doing before it (e.g., during a voice chatting the notification is temporarily ignored), the context in which the user was involved (e.g., if the user was at work, she used notifications to keep in contact with everything she did) and obviously on her habits.

This paper presents the architecture of a smart notification system that uses machine learning algorithms to manage incoming notifications according to context awareness and users habits. Such a system is composed by different modules that monitor environment and users to provide updated information to the core component, a central *Decision maker* module, that makes decisions about who should receive the incoming notification, on which device(s), in which moment and with which mode(s) (e.g., vibration, sound, light signal).

As a first step in building the described system, three different supervised machine learning algorithms were applied on a reduced dataset to establish the best device on which each incoming notification should be delivered. The dataset was made by a mixture of synthetic and real data taken from the MIT Media Laboratory Reality Mining project [3].

The remainder of the paper is organized as follows: Section 2 analyses existing related works, Section 3 describes the proposed architecture, Section 4 analyses the first developed prototype and reports the preliminary results obtained by using three different machine learning algorithms (Support Vector

Machine, Gaussian Naive Bayes, and Decision Trees) and Section 5 concludes the paper with some considerations and future works.

II. Background and Related works

The problems related to intrusive, annoying and repetitive notifications have been treated in several existing works and projects and several specific solutions can be found in literature. However, only a few of them uses machine learning to approach the problem.

Machine learning is based on algorithms able to learn from and make predictions on data: they use sample inputs, called training sets, to build a model that is then exploited to make predictions [4]. Two different main machine learning algorithm categories can be identified: supervised learning and unsupervised learning algorithms. The difference between them is in the used dataset: supervised learning algorithms use data that have already been classified and to which labels have already been assigned. This kind of data are called labeled data and helps the algorithm to make the same prediction in similar situations. Unsupervised learning algorithms, instead, try to find hidden structure in unlabeled data by creating groups of data with similar properties.

One of the work that addresses the overwhelming notification problem is the one proposed by Bohmer et al. [5] that explains how a smartphone abrupt full-screen notification used to alert user(s) about incoming calls forcibly interrupting whatever activity the user was already engaged in. They propose a smaller partial-screen notification as a solution to the problem. Moreover, Ardissono et al. [6] propose a notification model to reduce the disruptive effect of notifications on user attention. Their model predicts user activities using information received by different heterogeneous Web applications and uses these predictions to decide whether to show, postpone or delete received notifications. Even though these works address only specific problems connected to notifications, they are strictly related to our system and their notification modalities could be integrated in our future works.

In addition, some other projects have already developed works that could inspire or may be integrated in the next versions of our system. Roecker et al. [7], for instance, explore alternative approaches and strategies for email filtering and notification with the rationale of developing an unobtrusive notification interface that can be adapted to the users context. Leonidis et al. [8] present a semantics-based, context-aware notification system that provides personalized university alerts (e.g., reminders about timetable) to graduate students based on their preferences. Banerjee et al. [9] describe a universal notification system that considers connected solutions, context awareness, and user preferences to generate and distribute smart city notifications. Etter et al. [10] propose the Awareness and Notification Service (ANS) that makes applications aware of context changes by notifying them with appropriate rendering of intensity. Furthermore, some commercial products have already been developed: one of them is Google Inbox [11] that scans user email accounts to identify important and similar information aiming at presenting what it considers the most important parts of the email first and group similar emails.

Moreover, Poppinga et al. [12] conducted a large-scale, longitudinal study, collecting 6,581 notifications from 79 different users over a 76-day time period, aiming at developing a model for predicting suitable moments for issuing notifications. Unfortunately, this collected notifications dataset is not useful for our purposes: it contains only information about users reaction to a notification (immediately answer or not) without any information about the type of received notification and/or the user that received it.

Finally, a notification framework architecture was proposed by Arlein et al. [13]: it is similar to our system, in fact it allows notifications to be efficiently and effectively distributed and displayed in diverse new environments. However, it is different from our work due to a different approach to the problem. The first assumption that they do is related to the replication of the system: they assume that the system should be replicated for each user considering only devices in the decision process instead of the combination of both users and devices. Moreover, even though both the architectures accept notifications over multiple protocols, the architecture proposed by Arlein uses a deterministic approach to make decisions: a sequence of conditions are evaluated and, consequently, the system does not learn behaviors from previous detections (like we do using machine learning algorithm). In addition, they do not consider context aware and user habits: contexts are only handled in the adaptation process in which the system chooses to adapt the notification as a summary or a complete notification.

III. Architecture

The designed smart notification system is a modular system able to manage notifications using machine learning algorithms.

A typical intended usage of the system is described in the following scenario, that will be used as a running example: a user (let's call her Maria) is in her house and it is 10 o'clock in the morning. She owns a smartphone, a smart washing machine, a smart fridge and a smart TV. While Maria is having a shower, the washing machine ends its washing cycle and sends a notification to Maria. Considering that she is involved in an activity that does not let her use any available device, the system decides to postpone the notification of the message by 15 minutes, the predicted moment in which she will exit the bathroom and take her smartphone. Furthermore, due to the current time and the available notification modes, a sound and a message on the phone are chosen as methods to deliver the notification. Therefore, the smartphone is chosen as the receiver of the notification and the end of the shower is chosen as the right moment to notify it with a sound and a message on the phone.

Figure 1 shows the architecture of the proposed system: it accepts notifications from different external sources through its *Collector* module. Notifications are then sent to the *Decision maker* that is aware of environment status (e.g., weather information, current date and time), user context (e.g., location, status, current activity), and user habits and uses them to choose the best devices and the best ways (e.g., vibration, sound, or a light signal) to present the received notifications. Finally, the *Dispatcher* adapts the notifications to the chosen target devices and actually sends them.

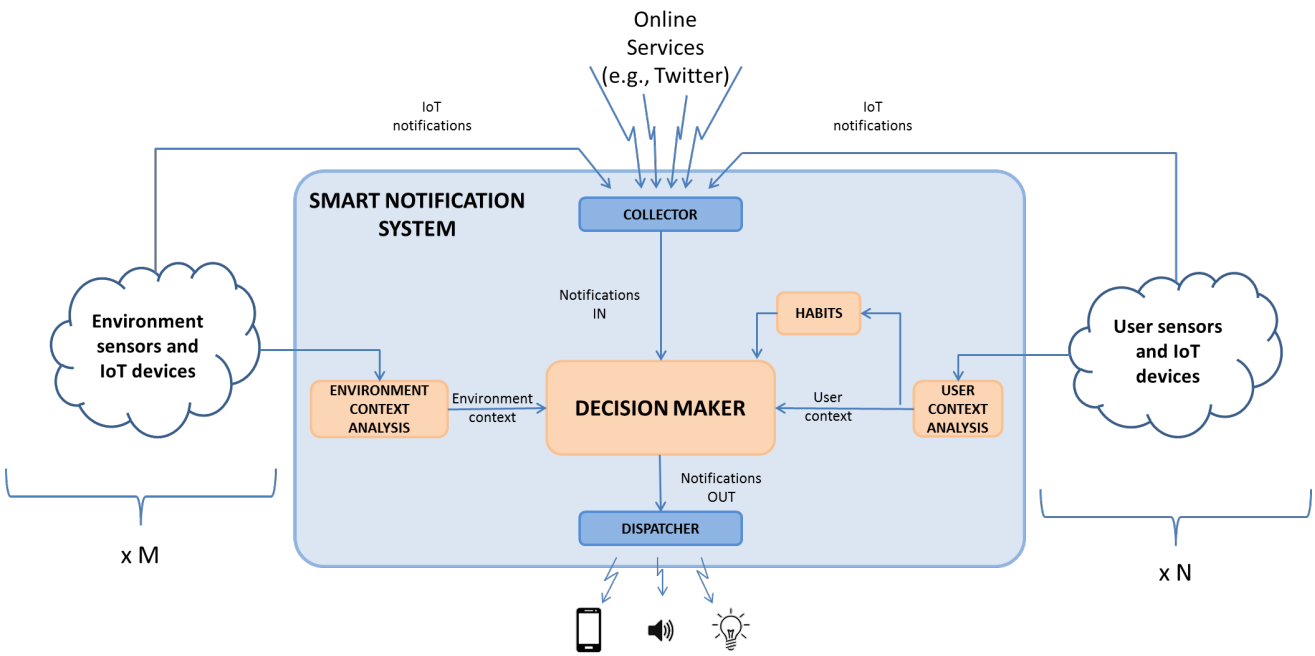

Fig. 1: Architecture design

Three main different sources are responsible for providing information about context:

- user personal sensors, devices, and services provide information related only to people (called "user context");
- environment sensors, devices, and services provide information related only to the ambient (called "environment context");
- general IoT sensors, devices, and services that provide information related to both users and environment (called "IoT devices").

The following description depicts the details of the most important architectural elements.

The ***Environment Context Analysis*** and ***User Context Analysis*** modules support the *Decision maker* for retrieving information shown in Table I about environment and user context. These data are acquired from external blocks such as sensors, cloud services (e.g., calendar), and IoT devices.

The ***Decision maker*** is the core of the system being responsible for all the decisions: every time it receives a notification, it takes the information from the two context analysis components and from the habits module, and decides:

- who should receive the incoming notification (one or more users);
- what is the best moment to show the notification to the chosen user(s) (i.e., a notification can be postponed or ignored if it does not contain important information);
- on which device(s) the chosen user(s) will receive the notification;
- which is the best way to notify the incoming notification.

The most important characteristic of this module is the way used to make decisions: a supervised learning classification algorithm is responsible for all the described choices.

User context	Current activity
	Current location
	User preferences
	Personal user information
Environment context	Current timestamp
	Other static information (e.g., building information)
Available IoT devices information	Owner
	Current location
	Current status (e.g., on, off, standby)
	Available modes

TABLE I: Context information

Information about incoming notification	Sender
	Receiver
	Type of notification
	Timestamp of receipt
	Contained message/information
Assigned labels to outgoing notifications	Target devices
	Chosen mode
	Delivery timestamp

TABLE II: Notification information

As already explained, supervised machine learning algorithms are trained through labeled datasets, so we defined the essential characteristics and information that the system should manage to make the right decision.

Table I shows the list of context information needed by the

algorithm to make decisions: information related to available IoT devices (such as device owner, device current location and device current status), user context and environment context (e.g., user current position or current timestamp). Table II, instead, lists the information related to both incoming and outgoing notifications. Every time a notification arrives, the *Decision maker* collects available context data and, using the trained algorithm, considers incoming information to make a decision.

Looking at the scenario presented at the beginning of the section and to better understand the behavior of the *Decision maker* a reduced dataset (i.e., showing key information only) is presented:

- User context information
  - user: Maria
  - current activity: having a shower
  - current location: house - bathroom
- Environment context information
  - current timestamp: 2015-06-11T10:00:30
- Available IoT devices information
  - Smartphone
    - current location: bedroom (home)
    - current status: on
  - Smart TV
    - current location: kitchen (home)
    - current status: off
- Information about incoming notification
  - Sender: washing machine
  - Receiver: every family member
  - Type of notification: service message
  - Timestamp of receipt: 2015-06-11T10:00:15
  - Message: washing cycle ended.

The *Decision maker* will use this dataset to make the following decision (the following information will be contained in the outgoing notification):

- target devices: smartphone
- chosen mode: sound + message
- delivery timestamp: 2015-06-11T10:15:30.

Moreover, as depicted in Figure 1, the *Decision maker* is supported by other modules. The first one is the ***Collector*** that is responsible for retrieving any kind of notification generated or arrived from every external service and IoT device.

Secondly, the ***Dispatcher*** module is responsible for the distribution of the notifications: it adapts the notifications generated by the *Decision maker* according to the output methods supported by each device and to the method selected by the *Decision maker*. Then, it sends them to the right device.

IV. Prototype and preliminary results

In order to evaluate the best machine learning approach applicable to managing overwhelming notifications, a preliminary version of the *Decision maker* module was prototyped and tested using a reduced dataset.

Aiming at obtaining data for the training and the test phases of the developed machine learning algorithm, the MIT Media Laboratory Reality Mining project [3] was chosen as the basis dataset containing as much needed information as possible with a consistent number of samples. MIT researchers used mobile phones to collect data about call logs, Bluetooth devices in proximity, cell tower IDs, application usage, and phone status. 94 people over 9 months were monitored and the collected data were, then, used to infer different user information including location.

Information	# of elements	Used values
senders	4	/
users (possible receivers)	94	/
personal devices per user	2	/
overlapping activities	7	working, sleeping, cooking, break from work, having a shower, programming, surfing the Internet
notification types	5	social network, security alarm, personal health, temperature notification, wearable fitness tracker notification
available user locations	3	work, house, elsewhere

TABLE III: Dimension of used dataset

Thus, a simplified dataset was created using a Python script that added synthetic information to the dataset. It contains the following data:

- Sender
- Receiver
- Type of notification
- Timestamp of receipt
- User current activity
- User current location
- Available devices for the user
- Target device.

For the purpose of training, a label was programmatically assigned to each sample representing the chosen device: Table III shows the dimensional details of the used dataset containing 165,289 samples.

Moreover, in this preliminary implementation we decided to simplify the work of the *Decision maker* by:

- choosing only one device as receiver of the notification instead of more than one;
- assuming that each device has only one available mode to show/notify the notification;
- ignoring the decision related to the best time to deliver the notification.

As already discussed, machine learning algorithms learn from and make predictions on data and the difference between labeled and unlabeled dataset was already explained, however another distinction between input data is needed. Considering that machine learning algorithms accept only numerical values as input and that these numbers can be related or not, we can divide input data into:

- *related data*, that is represented by an ordinal value [14], possessing the properties of ordering and proximity (e.g.,

ML Algorithm	Percentage of corrected predictions with unrelated data	Percentage of corrected predictions with related data
Support Vector Machine	81,6%	96.1%
Gaussian Naive Bayes	51.3%	83.4%
Decision Trees	99,9%	93.9%

TABLE IV: Percentage of corrected predictions obtained with used algorithms

if a machine learning algorithm accepts the receipt time of a notification as input, each value will be represented by a related number and so 1 o'clock is nearer to 2 o'clock than to 4 o'clock);

- *unrelated data*, that is data in nominal scale, i.e., data with equality property and without the ordinal and proximity ones (e.g., if a machine learning algorithm accepts the sender of a notification as input, each sender will be identified by a number and it is easy to understand that the first sender does not have any relation to the second).

In our situation all the information contained in the simplified dataset can be considered as unrelated data except from the receipt of timestamp

Each machine learning algorithm deals differently with related and unrelated data, so we decided to compare three different machine learning algorithms for this preliminary implementation: Support Vector Machine (SVM) [15], Gaussian Naive Bayes (GNB) [16] and Decision Trees (DT) [17]. The first two algorithms try to find a correlation between inserted data in order to classify them and so they work better with related data. Instead, the DT algorithm tries to create a flowchart-like structure in which each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a label. The paths from root to leaf represents classification rules and, as can be understood, it does not look for relations between data.

Moreover, two different experiments were conducted with each algorithm, one with only related data and another one with both related and unrelated data: we expected that SVM and GNB work worse with unrelated dataset than with related one and that DT works better with unrelated than with related.

The three algorithms were implemented using the Python programming language through the Anaconda distribution [18] and the Scikit-learn tool [19]. We used 80% of the data as training dataset and the other 20% for tests over 165,289 samples.

A notebook with the following characteristics was used for running all the experiments:

- CPU: Intel Core i7-4800MQ
- RAM: 16GB PC3-12800 (800MHz)

Table IV shows prediction accuracy results obtained with both related and unrelated: as expected, SVM and GNB seem to work worse than DT with unrelated data. But if we look at the results obtained with related one, we can easily establish that the SVM algorithm becomes the best algorithm as a consequence of the addition of related information. Moreover, the analysis of each wrong decision, we noticed that all the errors made by DT in the second experiment were related to combination of attributes that were not present in the training set, implying that DT do not work very well with unknown notifications contained in related data.

ML Algorithm	CPU time with unrelated data	CPU time with related work
Support Vector Machine	5296.1 s	5801.1 s
Gaussian Naive Bayes	1.2 s	12.9 s
Decision Trees	15.5 s	13.9 s

TABLE V: CPU time for a training phase with 33058 samples

ML Algorithm	CPU time with unrelated data	CPU time with related work
Support Vector Machine	40.19 ms	40.22 ms
Gaussian Naive Bayes	0.29 ms	0.31 ms
Decision Trees	0.001 ms	0.001 ms

TABLE VI: Average CPU time for each notification classification

The CPU time was collected and the obtained values are reported in Table V and Table VI[1]. Confidence interval are not present in the tables since all the calculations occur for a single run of each ML algorithm. As can be seen, the SVM algorithm is the slowest algorithm between the analyzed ones, however, the classification time is acceptable for real situations. Consequently, considering the accuracy percentage obtained with related data (that are the most complete) and the obtained CPU time, SVM is the most promising machine learning algorithm for notification classification and so it will be at the core of our future work. In addition, we will also evaluate the possibility of using it in combination with other algorithms to obtain better predictions with both related and unrelated information.

V. Conclusions

In this paper, we present a modular architecture that uses machine learning algorithms to manage incoming notifications. According to context awareness and user habits, the presented architectural system is able to decide who should receive the incoming notification, on which device(s), in which moment and with which mode(s) (e.g., vibration, sound, light signal). A simplified version of the architecture has been prototyped and a preliminary validation has been performed. Three different machine learning algorithms were used on an adapted dataset and results show that the SVM algorithm is the most promising algorithm for our purposes in terms of prediction accuracy and, so, it will be at the core of our future efforts.

Future work will extend the dataset to include all the information discussed in the Architecture section and a system to collect real data and real notifications from volunteers will be developed. Furthermore, a careful evaluation of the machine learning algorithms will be performed and the possibility of using different algorithms after other ones will be evaluated. Finally, the prototype will be enhanced to include unconsidered

[1]The time reported in Table VI is computed dividing the execution time for classifying all the notifications for the number of notifications.

blocks and a more complete implementation will be evaluated by an adequate number of people aiming at obtaining structured information about the accuracy of predictions made by it and a more completed dataset for the system training.

ACKNOWLEDGMENT

Teodoro Montanaro currently exploits a research grant by Telecom Italia SWARM Joint Open Lab.

REFERENCES

[1] K. Church and R. de Oliveira, "What's up with whatsapp?: Comparing mobile instant messaging behaviors with traditional sms," in *Proceedings of the 15th International Conference on Human-computer Interaction with Mobile Devices and Services*, ser. MobileHCI '13. New York, NY, USA: ACM, 2013, pp. 352–361.

[2] A. Sahami Shirazi, N. Henze, T. Dingler, M. Pielot, D. Weber, and A. Schmidt, "Large-scale assessment of mobile notifications," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '14. ACM, 2014, pp. 3055–3064.

[3] N. Eagle and A. (Sandy) Pentland, "Reality mining: Sensing complex social systems," *Personal Ubiquitous Comput.*, vol. 10, no. 4, pp. 255–268, Mar. 2006.

[4] P. Harrington, *Machine Learning in Action*. Manning Publications Company, 2011.

[5] M. Böhmer, C. Lander, S. Gehring, D. P. Brumby, and A. Krüger, "Interrupted by a phone call: Exploring designs for lowering the impact of call notifications for smartphone users," in *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI '14. New York, NY, USA: ACM, 2014, pp. 3045–3054.

[6] L. Ardissono, G. Bosio, A. Goy, G. Petrone, and M. Segnan, "Managing context-dependent workspace awareness in an e-collaboration environment," in *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 03*, ser. WI-IAT '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 42–45.

[7] C. Roecker, V. Bayon, M. Memisoglu, and N. Streitz, "Context-dependent email notification using ambient displays and mobile devices," in *Active Media Technology, 2005. (AMT 2005). Proceedings of the 2005 International Conference on*, May 2005, pp. 137–138.

[8] A. Leonidis, G. Baryannis, X. Fafoutis, M. Korozi, N. Gazoni, M. Dimitriou, M. Koutsogiannaki, A. Boutsika, M. Papadakis, H. Papagiannakis, G. Tesseris, E. Voskakis, A. Bikakis, and G. Antoniou, "Alertme: A semantics-based context-aware notification system," in *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, vol. 2, July 2009, pp. 200–205.

[9] S. Banerjee and D. Mukherjee, "Towards a universal notification system," in *Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2013 IEEE/WIC/ACM International Joint Conferences on*, vol. 3, Nov 2013, pp. 286–287.

[10] R. Etter, P. Costa, and T. Broens, "A rule-based approach towards context-aware user notification services," in *Pervasive Services, 2006 ACS/IEEE International Conference on*, June 2006, pp. 281–284.

[11] "Google inbox is an organized place to get things done and get back to what matters. bundles keep emails organized." [Online]. Available: http://www.google.com/inbox/

[12] B. Poppinga, W. Heuten, and S. Boll, "Sensor-based identification of opportune moments for triggering notifications," *Pervasive Computing, IEEE*, vol. 13, no. 1, pp. 22–29, Jan 2014.

[13] R. M. Arlein, S. Betge-Brezetz, and J. Ensor, "Adaptive notification framework for converged environments," *Bell Labs Technical Journal*, vol. 13, no. 2, pp. 155–159, Summer 2008.

[14] S. S. Stevens, "On the theory of scales of measurement," *Science, New Series*, vol. 103, no. 2684, pp. 677–680, Jun. 1946.

[15] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995. [Online]. Available: http://dx.doi.org/10.1007/BF00994018

[16] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Pearson Education, 2003.

[17] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, Mar. 1986.

[18] "Anaconda is a completely free python distribution that includes over 195 of the most popular python packages for science, math, engineering, data analysis." [Online]. Available: https://store.continuum.io/cshop/anaconda/

[19] "Scikit-learn is a simple and efficient tools for data mining and data analysis, accessible to everybody, and reusable in various contexts and built on numpy, scipy, and matplotlib." [Online]. Available: http://scikit-learn.org/stable/