Active Data Collection of Health Data in Mobile Devices

Ana Rita Bamansá Siles Machado

December 2021

Abstract

This paper aims to develop an intelligent notification system to help sustain user engagement in mHealth applications, specifically those that support self-management. The proposed approach focuses on applying Reinforcement Learning (RL) in its more simplistic format, intending that the developed agent learns what moments are the most opportune for each user, throughout their day, only from easily obtainable non-sensitive data and usage history. This history allows the agent to remember how the user reacts or has reacted in the past to its actions. Various formulations of the problem at hand are designed, developing several alternatives for the components required by the different implemented RL algorithms. Through the permutation of these components, the most desirable combination of initialization method, algorithm, state representation, and reward definition is searched. A simulator was developed to mimic the behavior of a typical user and utilized to test all possible combinations with users experiencing distinct lifestyles. This work showed promising results and, although still requiring further testing to be fully validated, demonstrated that an efficient and well-balanced notification system can be built with simple formulations of an RL problem and algorithm. Furthermore, it proved that there is no absolute necessity for this type of system to require the access and utilization of sensitive user data. This approach diminishes privacy issues that might concern the user and limits sensor and hardware concerns, such as lapses in collected data or battery drainage.

Keywords: mHealth; Notifications; Machine Learning; Personalization; Reinforcement Learning; Receptivity.

1. Introduction

Mobile health, or mHealth, is defined by the World Health Organization (WHO) as "medical and public health practice supported by mobile devices" such as phones, wearables or other patient monitoring devices [16] and is a great vehicle for the support of self-management in Noncommunicable diseases (NCDs).

NCDs, also known as chronic conditions [3], such as cancer, diabetes, stroke, and other chronic respiratory or cardiovascular diseases, are the leading causes of death and disability worldwide. These represent more than 70% of all deaths and create devastating health consequences. This epidemic threatens to overwhelm health systems across the world, making it essential for governments to prioritize health promotion and disease management [4]. The ability for patients to employ self-management is now more vital than ever and many studies have already shown promise for its application in helping manage these chronic conditions [1]. However, some key factors still restrict the adoption of mHealth, for instance, the lack of standards and regulations, privacy concerns, or the limited guidance and acceptance from traditional healthcare providers. The decisive factor, however, is always the necessity for widespread user adoption and engagement [14].

Phone notifications are widely employed to achieve user engagement, having been proven to significantly increase verified compliance when compared with identical trials that did not employ this technique [2]. Nonetheless, the risk of intrusiveness into daily life is imminent. Furthermore, consumers are known to highly dislike excessive or inopportune notifications, primarily when originated by machines [8]. For these reasons, mHealth applications must function and communicate without burdening the consumer. Hence, this paper focuses on developing an intelligent notification system that intends to increase continued engagement by helping applications communicate with users when they are receptive, not bothering them on inconvenient occasions.

2. Background

The purpose of this paper is to develop a mechanism that is able to identify opportune moments for notifications. This goal entails challenging cross-disciplinary subjects such as information technology, medicine, and psychology, which plays a big part in understanding human behavior.

Push notifications emerged to contrast with the traditional pull-based information retrieval and delivery paradigm, where the user would have to be the one to initiate a request. Throughout an average day, users can get over 50 notifications on their phones. Hence, feeling overwhelmed or experiencing growing negative sentiments towards individual notifications or apps is expected [8]. Grounded in previous studies, receptivity is defined as the degree to which a user considers that a notification is received at an opportune time. Currently, there is yet no systematic way to infer user availability and receptivity. Identifying ideal moments for interaction with a permanently high level of accuracy is a complex problem due to its dependency on many aspects of a user's context [8], such as location, movement, time or psychological state.

Nearly all current applications employ a basic interaction model that assumes the user is available and willing to engage with the device at any time or at specific predefined time schedules. However, it can be argued that the pervasive nature of phones and their inherent constant accessibility does not necessarily translate into receptivity. The potentially disruptive impact of these applications should be compensated with the customization of notifications' characteristics (such as presentation or alert type) [9], content [8], and an intelligent approach to deliver them [7]. Therefore, systems that attempt to handle such notifications intelligently are increasingly relevant. Although many studies have been published on this kind of system, most of the existing prior work can be divided into [5]: detecting transitions between activities, assuming these represent the most opportune timings in a user's routine, or inferring receptivity from the user's context.

Notification systems that learn how, when, and where people engage with their phones allow for personalized communications, enhancing user experience. Systems such as these have resorted primarily to machine learning (ML) techniques due to their capability of discovering patterns in data [8]. ML techniques can be divided into five: Supervised (SL), Unsupervised, Semi-Supervised, Active and Reinforcement learning (RL). SL is the most common for this kind of smart system, presumably due to the tendency of seeing the problem at hand as the need to classify users, their preferences, or even labelling moments as opportune or inconvenient. Albeit its known capacity for swift adaptation to dynamic, complex environments, RL is more complicated to apply than SL, leaving it with few implementation attempts in the mHealth field [11]. Nonetheless, its ability to learn in real-time while attempting to maximize user satisfaction and not requiring the existence of a training dataset makes it a desirable study subject. In this method, an agent learns through interactions with the surrounding environment, resorting to a framework that defines them in terms of states, actions, and rewards. In order to explore the potential of employing RL, in this work, the developed notification system is formulated as a RL problem.

3. Methodology

This paper envisions the search for opportune notification timings as a reinforcement learning problem. Through the application of RL, our system aims to learn user preferences, routines, and habits merely from notification interaction data. Here, the environment is implemented by a simulator that generates user responses following a probabilistic model. The development of the simulator is explained in further detail in section 3.4.

Our work is centered around mHealth and improving continued user engagement. Hence, the main aim is to discover one moment throughout the day when the users are available and willing to answer a notification that leads to the required action. This action could be any self-management task required by any mHealth application. When this goal is achieved, no more notifications should be sent. For that reason, the RL agent considers an answered notification as a terminal state, meaning that the episode, in our context, a day, has ended and that the agent only starts working again when the next day begins. The only other terminal state occurs at the end of the day (24h). Tasks such as this are called episodic tasks [12].

The main objective is that, through observing the user and the environment's state, the agent should decide whether to send a notification or stay silent. Then, if a notification is sent, the agent observes the user's reaction and continually learns from it. In our case, accepted notifications denote positive signals while dismissed or ignored ones are seen as negative reinforcement signals, penalizing the agent for the action taken. The agent's behavior changes accordingly, always intending to increase the long-run sum of rewards (reinforcement signals). These components allow the creation of a sense of cause and effect, the existence of explicit goals, and help the agent deal with the environment's uncertainty and nondeterminism. A goal is expressed by the reward structure, which, as previously mentioned, is to get the user to answer one notification per day. Through this framework of states (S), actions (A), and rewards (R), in this approach, as in most RL methods, the intention is to learn the optimal action-value function (Q), which can be defined as the expected empirical reward for each action. By learning Q, the agent knows which action generates a higher reward, knowing the best action for each moment [12].

The best combination of algorithm, rewards and state definitions must be found to discover the most efficient solution for this learning problem. For that reason, this work reviews, selects, and tests several combinations.

3.1. RL Algorithms

In order to perform experiments that are as varied as possible, several consensually recommended algorithms were implemented.

Upper Confidence Bound (UCB) - UCB emerges as a widely accepted nonassociative bandit[1], as it considers the problem as only a single state. UCB achieves exploration by subtly favoring the selection of actions that have the potential to be optimal and have been employed the less [12]. To do so, it applies the selection rule:

$$A_t \doteq \underset{a}{argmax} \left[Q_t(a) + c\sqrt{\frac{\ln(t)}{N_t(a)}} \right] \tag{1}$$

where A and a represent actions, t represents the current timestep, Q is the action-value function and c, the confidence level that controls the degree of exploration. Finally, $N_t(a)$ represents the number of times action a has been selected prior to time t.

In (1), the term inside the square root represents the uncertainty of the estimates of action values, making A_t an upper bound of the probable value of each action. UCB ensures that, given enough time, all actions are eventually performed. Hence guaranteeing that the agent explores the action space properly, learning the best action for the problem it is dealing with. As time goes on and different actions are performed, to each, the sum of received rewards and the number of selections are associated. With these values, the action-value function Q is updated at each timestep.

It should be noted that since UCB is a single state algorithm, it learns what is better suited for that state only, which in this paper's context would be very restrictive. Hence, this algorithm is applied in two manners that are expected to provide efficient results by leveraging its advantages but still considering that the user goes through different states throughout time. Firstly, UCB Day, where a different UCB instance is applied to each hour of the day, allowing the agent to learn what action is better suited for each decision point, which in our case is hourly. UCB Week is the second, more personalized approach, where a separate instance is applied to each hour of each day, each day of the week. Considering that the week has 7 days, and 24 instances are created for each day, UCB Week combines 168 instances, learning what is the most beneficial action for each decision point, according to the weekly routine of a user.

Tabular Q-learning (TQ) - Tabular methods are settled on the core idea of RL in its most straightforward format, that both the state and action spaces are small enough for the estimates of action values and their mapping with specific contexts to be represented as arrays or tables. These methods are commonly used due to their simplicity and ease of computation. Q-learning was initially defined by Watkins in [15] and can be defined by the following expression:

$$\begin{aligned} Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \\ \gamma \max_a (Q(S_{t+1}, a)) - Q(S_t, A_t)] \end{aligned} \tag{2}$$

where A and a represent actions, S represents the state and R the reward. Additionally, t represents the current timestep, Q is the action-value function for each state-action pair, α is the learning rate and, lastly, γ is the discount factor. The learning rate, α, determines when Q-values are updated, overriding older information. The discount factor, γ, models the relevance of future rewards by causing them to lose their value over time so that more immediate ones are valued more highly.

As seen above, (2) maximizes the next state-action pair, meaning that the policy is greedy. It does not take into account how probable each action actually is under the current policy. Unlike similar approaches, this algorithm is an off-policy method, meaning that, although not true, it assumes a greedy policy is being used. That can be verified in (2), where it is visible that Q is updated using the value of the following state and the value of the greedy action a, instead of the value of the real action taken. However, different policies can be applied to actually choose the desired action. This choice should take into consideration the context and particularities of the problem in question. In our work, the ε-greedy policy, where the agent behaves mostly in a greedy way but occasionally, and with a small probability ($\varepsilon > 0$), selects a random action, showed promising results and was henceforth applied.

Deep Q-Learning (DQN) with Experience Replay - Developed by Mnih et al. in [10], the Deep Q-learning agent, combines the previously described Q-learning algorithm with a Neural Network(NN). This network is usually a deep convolutional NN due to its many layers and fully connected network. Here, the agent's brain is the NN instead of a table or array. It receives an observation and outputs the estimated values for each of the available actions. It is updated through the mean square error loss function, where the difference between the current predicted Q-values (Q_θ) and the true value (Q_{target}) is computed according to:

[1]Bandit problems are RL problems where the agent learns to act in a single state setting, not requiring an association between actions and states (nonassociative).

$$Q_{target}(t) = \begin{cases} r_t, \\ for\ terminal\ \phi_{t+1} \\ r_t + \gamma \max_{a'}(Q_\theta(\phi_{t+1}, a')), \\ for\ non-terminal\ \phi_{t+1} \end{cases} \tag{3a}$$

$$Loss(\theta) = \sum (Q_{target}(t) - Q_\theta(\phi_t, a_t))^2 \tag{3b}$$

where a represents an action, ϕ represents the state and R the reward. Additionally, t represents the current timestep, Q is the action-value function for each state-action pair, and θ represent the network weights.

While this type of NN allows for more flexibility, it sometimes comes at the cost of stability. For that reason, many extensions of this algorithm have already been designed and tested. One, in particular, is called Experience Replay [6], where the agent memorizes the state, action, and effect of that same action in the environment for every timestep. After completing an episode, it replays the gathered experiences by randomly selecting a batch of a particular size and training the network with it. This replay helps reduce instability produced by training on highly correlated sequential data and increases the learning speed.

3.2. State Representation

Ideally, a notification system would have access to the current activity, emotional state, location, and other private information that we prefer not to utilize to minimize privacy issues. Here, the focus is on using accessible information such as the time or day of the week, the user's reaction to notifications, or the number of notifications already answered. Hence, the aim is to demonstrate that efficient results can be obtained from more simplistic representations of a user's state. Thus, for the DQN and TQ algorithms, the state representations below described were designed.

Four representations are contemplated: S1 and S3 have similar formats, both containing the time of the day in minutes, the number of notifications already sent and answered that day, and the last user reaction. The difference between these states is that S1 also contains the day of the week, depicted by values from 0 to 6, allowing for a representation of a weekly routine instead of a simple daily routine such as S1 permits. S2 and S4 were born from a similar approach, both containing an array of 24 elements where all positions start as 0 and then, throughout the day, each element may change depending on the outcome of the action chosen at every hour (below, the numbers used to express each user reaction are described). S2 has an additional element which, again and with the same aim as before, represents the day of the week.

The developed states resort to easily obtainable information, focusing primarily on knowing how far the agent is from its objective and how the user reacts to its actions. Furthermore, since the generic goal is to learn the most opportune timings, time and, in some cases, even the day of the observation are also tracked. With the purpose of recording users' reactions to the actions of the agent, 3 options were defined and associated with a value: 0, meaning that in the last timestep, a notification was sent and ignored or dismissed; 1,a notification was sent and positively addressed ; 3, a notification was not sent.

3.3. Reward Definition

At each timestep, the received reward is a scalar – $R_t \in \mathbb{Z}$. The efficiency of any RL application is dependent on how accurately these numbers frame the agent's goal and assess progress in reaching it. Hence, designing reward signals is one of the most critical tasks in implementing an RL technique. In the case of games with predefined scores, the definition of different rewards is straightforward, but some cases involve goals that are more challenging to translate into signals. Furthermore, it is not uncommon that, when rewards are poorly structured, agents find unexpected ways to get higher rewards, which might be undesirable.

The final selection of rewards that were applied during our experiments are structured in the following manner: when a notification is sent and the user does not answer, the agent receives reward a. However, if the user responds then the received reward value is b. Contrarily, if a notification is not sent the agent receives c. Lastly, if the episode, in this context a day, ends without having achieved the goal of one answer then d is received. Thus, the rewards assume values in the set $R = \{a, b, c, d\}$. We define the following alternatives for the values of $\{a, b, c, d\}$: R1 = $\{-1, 2, -1, -2\}$; R3 = $\{-2, 2, -1, -3\}$; R5 = $\{-2, 2, 0, -3\}$; R6 = $\{-3, 2, 0, -3\}$.

The general idea we wish to transmit to the agent with these structures is that the goal is to get the user to answer one notification without bothering them by sending notifications that go unanswered.

3.4. Environment Model

In our experiments, users are simulated by combining underlying models, a behavior model and a response model, further detailed in the following sections. The creation of this simulator allows us to mimic the daily routine of a user. Depending on the algorithm, previous user responses, or any other initial information, the agent decides whether to send a notification or remain silent. When it decides that a notification should be sent, the simulated user is the one who determines if it responds or not. This work does not differentiate between ignoring the message or explicitly dismissing it, considering both as "No Answer" received. This approach is followed because we do not wish to understand why a moment is less opportune but simply that it is. In this way, the users' answers or lack of it are registered, and their motivations disregarded. Furthermore, the user's answer is considered to be either immediate or non-existent.

3.4.1 Behaviour Model

This model reflects a users' routine, for example, the activities performed, their duration, and the user's location. It mirrors the ExtraSensory dataset [13], which aggregates daily traces of 60 participants. Measurements from smartphones and smartwatches were collected, along with self-reported labels. Since this data was collected in the wild, its reliability is not perfect; after processed and cleaned, it considers 51 possible tags, shown in Appendix A (15 locations, 8 primary activities, 28 secondary ones). These include primary activities, which describe movement or posture and are mutually exclusive, and secondary activities, which represent a more specific context. For the latter, such as for locations, the user could apply several tags to a single instance in time. In this simulator, the users' state is represented as the combination of one primary activity and a set of up to 43 possible secondary tags, composed by secondary activities and locations.

From the available data, three user traces were chosen. These were selected according to two main concerns: providing lifestyles as distinct as possible while ensuring the availability of enough data to represent a week in these users' lives.

3.4.2 Response Model

The response model simulates how a user responds to a notification in any given context, originating the observations that our agent receives throughout the simulation. Based on our investigation, a set of behaviors that researchers consensually agree users tend to show were considered when implementing this model [8, 9, 7].

First, when the behavior model presents labels such as sleeping, which ensure an inability to answer, the simulator does not respond to notifications. Furthermore for labels that represent tasks such as driving or being in a meeting, for which usually a low probability of answering is associated, the simulator tends not to respond;

Second, a randomness level is always associated with every decision the simulator makes, except when the user is sleeping. This level intends to express the same randomness a human would show in their daily life;

Third, a component (β), defined as the exponential decay in (4), is used to convey the diminishing desire to use the app that most users would experience as the number of daily notifications rises.

$$\beta(n_t) = P(Answer \mid n_t) = e^{-\lambda n_t} \tag{4}$$

Here, n_t represents the number of messages already sent during the current day. λ equals 0.3, chosen to guarantee reasonable values are obtained.

Forth, each user has a predefined prior probability of answering $P(A)$ and not answering $P(\overline{A})$. This value represents a person's predisposition to be on their phone and regularly use a mHealth application. For this reason, these are fixed values selected depending only on how compliant or not our simulated user ought to be.

Assuming statistical independence between labels and following the Naive Bayes probability model (5), the probability of the user answering or not, given their current context, is calculated.

$$\begin{aligned}
&P(C \mid L_0, ..., L_i, \overline{L_{i+1}}, ..., \overline{L_{l_t}}) \\
&\propto P(C, L_0, ..., L_i, \overline{L_{i+1}}, ..., \overline{L_{l_t}}) \\
&\propto P(C)P(L_0 \mid C)...P(L_i \mid C)P(\overline{L_{i+1}} \mid C)...P(\overline{L_{l_t}} \mid C) \\
&\propto P(C)\prod_{j=0}^{i}[P(L_j \mid C)] \prod_{k=i+1}^{l_t} [P(\overline{L_k} \mid C)]
\end{aligned} \tag{5}$$

A set of L_t labels, provided by the behavior model represents this context. For every instant in time, there are i labels that describe the moment (L) and $(l_t - i)$ that were not chosen and indicate activities the user is not currently doing $(\overline{L})$. Considering our two possible classes (C), $Answer(A)$ and $NoAnswer(\overline{A})$, the model is formulated as presented in (5).

The values of $P(L \mid C)$ and $P(\overline{L} \mid C)$ are unknown. To calculate these, the Bayes' theorem, $P(L \mid C) = \frac{P(C|L)P(L)}{P(C)}$, can be applied. Additionally, knowing the conditional probability formula, $P(C \mid L) = \frac{P(C,L)}{P(L)}$, and $P(C,L)+P(C,\overline{L}) = P(C)$, it is possible to calculate $P(\overline{L} \mid C)$ using only $P(L \mid C)$. Leaving now only the values of $P(C \mid L)$ and $P(L)$ as unknown. Hence, these were transformed into either obtainable components from the behavior's model dataset or reasonably estimated.

Estimation of Conditional Probability Values: For each label provided by the behaviour model, reasonable values were defined for the probability of answering given that label ($P(A \mid L)$) and not answering given that same label ($P(\overline{A} \mid L)$). These values are detailed in Appendix A.

Calculation of the Probability of each Label: The labels, which are considered mutually independent, conditional only to C, are supplied by the dataset. From the latter, the probability of each label can be calculated according to the formula presented in (6).

$$P(L_k) = \frac{N_{L_k}}{N_L} \tag{6}$$

The result is dependant on the chosen user since N_{L_k} represents the number of times L_k occurs in their routine, and N_L defines the total number of labels in that same routine.

Final Response Probability Model: Now with the values of $P(A \mid L)$, $P(\overline{A} \mid L)$ and $P(L)$ known for each label, the before unknown values of $P(L \mid C)$ and $P(\overline{L} \mid C)$ are easily obtained. As previously stated, there is an additional element used to represent user discontentment with notification volume, β. That element is incorporated in (7) and (8), finalizing our expressions as,

$$P(A \mid L_0, ..., L_i, \overline{L_{i+1}}, ..., \overline{L_{l_t}}) \propto \beta \left[P(A) \prod_{j=0}^{i} [P(L_j \mid A)] * \prod_{k=i+1}^{l_t} [P(\overline{L_k} \mid A)] \right] \tag{7}$$

$$P(\overline{A} \mid L_0, ..., L_i, \overline{L_{i+1}}, ..., \overline{L_{l_t}}) \propto (1-\beta) \left[P(\overline{A}) \prod_{j=0}^{i} [P(L_j \mid \overline{A})] * \prod_{k=i+1}^{l_t} [P(\overline{L_k} \mid \overline{A})] \right] \tag{8}$$

For each instance, the above presented factors are estimated, normalized, and, resorting to a simple sampling method, the simulator's response is determined.

$$X \sim U(0,1)$$
$$\hat{c} \in \{A, \overline{A}\} \sim \underset{C \in \{A, \overline{A}\}}{\| P(C \mid L_0, ..., L_i, \overline{L_{i+1}}, ..., \overline{L_{l_t}}) \|} \leqslant X \tag{9}$$

We believe this sampling technique allows us to reflect the ambiguity of users more accurately. $\hat{c}$ represents the class that defines the simulator's response. Depending on the simulator's reaction and the state of the environment, the algorithm then obtains the respective reward and adjusts the strategy accordingly.

4. Experiments

4.1. Model Initialization Methods

One of the main objectives of this paper is to analyze the efficiency levels that algorithms can obtain when models are initialized in different manners.

No Previous Knowledge Models (Online Learning) - This method entails no previous experience, implying that the models start with no prior knowledge and learn only from interaction with a specific user. It is expected that this method would allow for a better customization and better results in the long run. However, it is also anticipated that it shall take longer to start doing so since it must learn every user preference from scratch.

Previously trained models (Offline Learning) - Here, models are trained with two different users before being tested with a third one, where they only apply what they have learned from previous experience. These models are expected to provide acceptable results from the moment they are deployed since they are likely to have already learned generic preferences that people share amongst different lifestyles. Nonetheless, since no learning happens after the training stage, these are not personalized and are not expected to reach such good results as the remaining methods.

Previously trained adaptive models (Combination of Offline and Online Learning) - In this case, models are likewise trained before being deployed. However, they continue learning, which allows them to start more efficiently than models with no previous knowledge while also growing to be customizable. Assuming the chosen users' routines are varied enough to provide generalized knowledge that could then be applied to any user, this model, which combines the two previous ones, is expected to offer the best and most stable results.

4.2. Users' Routine - Daily vs. weekly routine

This paper hypothesizes that patterns of opportune timings can be found in a person's routine. By applying the different state representations of Q-learning and DQN and the different formulations of UCB, this work intends to test if higher levels of efficiency can be obtained when letting the agent learn what a typical week is for the user instead of a typical day. It is expected that when modeling opportune timings throughout a week, the agent takes longer to learn, but if enough time is provided, better results can be obtained. It is essential that it can be understood if this extra time taken to train provides improvements that are considered significant enough for the user to accept the disturbance of a not yet so well-trained model for more extended initial periods.

4.3. Performance Metrics

As performance metrics of our algorithms, we selected two: Goal Achievement Rate (G_r) and Notification Volume (N_v).

$$G_r = \frac{N_A}{N_{Days}} \tag{10}$$

$$N_v = \frac{N_{Sent}}{N_{Days}} = \frac{\sum_{i=0}^{Days}(N_{A_i} + N_{\overline{A}_i})}{N_{Days}} \tag{11}$$

G_r, in (10), is defined as the fraction of accepted notifications (N_A) over the number of episodes being tested (N_{Days}, each episode representing a day). High G_r values show that our agent was able to identify when users are open to receiving and answering notifications throughout the day. However, an agent may increase the G_r by simply increasing interaction with users. Thus, the volume of sent notifications is also tracked to balance this effect. With (11), the average number of notifications sent per day, also referred to as notification volume, can be obtained.

A well-behaved agent presents a high response rate (G_r) while maintaining a low notification volume (N_v), ensuring in this way that our system gets a response without bothering the user when he is not receptive.

5. Results & Discussion

Experiments were performed for all combinations of the described initialization methods, algorithms, states, and rewards. For each, 3 tests were executed by applying the leave-one-out technique for a set of 3 simulated users. The respective median was then determined as a measure of central tendency to diminish the influence of outliers. All graphs presented show the average result among tested users, employing the N_v and G_r metrics. The standard deviation was also analyzed and is likewise depicted in the displayed graphs. Furthermore, in the tables shown throughout this section, the average G_r and N_v values obtained over 300 days of training are presented.

Analyzing these results is a complex task since users appreciate different aspects in systems such as the one here developed. Ideally, the notification system should have the following properties: first, G_r should approach 100% and N_v 1 notification per day. Standard deviation values should not be high, ensuring consistency when applying the models to different users and in various situations. Furthermore, the results should be stable, meaning that the provided service should be consistent, not showing significant, unexpected variations in the N_v and G_r values throughout the training process. Although the system is expected to explore, user experience should not be affected in such a way that users would start experiencing dissatisfaction towards the application. Lastly, from deployment, the agent should not require a long period to start providing an acceptable level of user experience. Users must be able to start seeing the benefits of their effort early on, reinforcing user engagement and commitment.

To provide a bit more clarity regarding the conclusions that can be reached from the obtained results, the average values over all simulations of the permutations of algorithms, states, and rewards in each initialization method are shown in Table 1. Additionally, the average values over all simulations of the combinations of initialization methods, states, and rewards in each algorithm are likewise shown. The analysis of these results is provided in the following sections.

—	G_r	N_v
No Previous Knowledge	0.887±0.017	3.049±0.785
Previously Trained	0.888±0.069	3.351±1.338
Previously Trained Adaptive	0.963±0.041	2.877±0.881
UCB	0.905±0.049	2.082±0.399
TQ	0.975±0.015	2.983±0.837
DQN	0.854±0.067	3.706±1.467

Table 1: Average values over all combinations of each initialization method and algorithm.

5.1. Previously Trained Model

These simulations confirmed that a generic prediction model trained on multiple users' data tends to be less accurate for predicting the interruptibility of a specific user than individual-based models. This statement can be confirmed in Table 1, where the average values obtained for each initialization method are stated. Furthermore, when compared to the other two initialization methods, this model displays, on average, a higher standard deviation. Nonetheless, it obtains satisfactory results when resorting to the right combination of algorithm, state, and reward, which in this case is: DQN, using state S1 and S3, and rewards R5 and R6, shown in Table 2.

DQN	G_r	N_v
S1 with R5	0.996 ± 0.005	1.926 ± 0.095
S1 with R6	0.983 ± 0.012	1.813 ± 0.123
S3 with R5	0.996 ± 0.004	1.931 ± 0.082
S3 with R6	0.992 ± 0.003	1.864 ± 0.121

Table 2: Previously Trained - DQN.

Here, the network can leverage less complex state representations throughout the training phase and learn generic user preferences better than any other combination. This shows that, if the purpose is to learn generic preferences, it should be done in the less detailed manner possible, which in this context is represented by modeling a nonspecific daily routine (S3). Although not adaptable to new users' routines, a consistently pleasant user experience can still be offered. However, if applied to a user that has atypical habits, this model would not prove satisfactory since, at its core, it is not learning specific user preferences and adapting to their schedule, but simply applying previous knowledge. Hence, this implementation resembles notification systems that apply supervised learning techniques, not exploring the potential benefits of RL.

5.2. Previously Trained Adaptive Model

As expected, and shown in Table 1, this method appears to provide the overall best performing average results amongst all three initialization techniques since it can be refined as the final user is actively using the application.

Both UCB and TQ present a good balance between our metrics, as shown in Table 3.

Algorithm	State/Reward	G_r	N_v
UCB Day	with R3	0.987 ± 0.014	1.721 ± 0.127
	with R5	0.987 ± 0.009	1.716 ± 0.009
TQ	S1 with R5	0.982 ± 0.013	1.907 ± 0.289
	S1 with R6	0.981 ± 0.009	1.949 ± 0.237
	S2 with R5	0.985 ± 0.004	1.945 ± 0.285
	S2 with R6	0.981 ± 0.011	1.922 ± 0.298

Table 3: Previously Trained Adaptive - UCB and TQ.

As seen in Figure 1, and as happens through most combinations executed in this initialization method, although N_v converges early on, G_r values do not, which signifies that the agent is still exploring and attempting to learn the user's most opportune timings. While doing so, and due to the consequent inconsistency in the provided notification service, this behavior may put at risk user engagement with the application.

5.3. No Previous Knowledge Model

Looking at Table 1, this method appears to be less efficient than the Previously Trained Adaptive model,taking longer to achieve efficient G_r values (2 to 3 months). When analyzing Figure 2 in more detail, it can be concluded that although the average values are worse than the ones detailed in the other initialization methods, that is caused by the model's lack of previous experience and knowledge. Meaning that in the beginning this model provides a worse user experience, and G_r and N_v results, due to the fact that it is learning every user preference from scratch. These initial results diminish the presented average G_r values while raising the average N_v values. However, in the long run, the personalized models achieve better results that converge faster than the ones shown in the Previously Trained Adaptive model. This convergence implies that users are provided with a more consistent experience due to the system's continued success at finding opportune times, hence maintaining a high G_r, and the lack of peaks in the N_v values, consistently and steadily approaching about two notifications per day as time goes by.

For this case, simple combinations of both UCB Day and the TQ algorithm, with state representation S1, provide the best results, shown in Table 4. Furthermore, it should be noted that the lower average deviation values are obtained with this implementation, as visible in Table 1.

Algorithm	State/Reward	G_r	N_v
UCB Day	with R5	$0.979 \pm 2.8\mathrm{e}^{-17}$	$1.714 \pm 2.8\mathrm{e}{-17}$
	with R6	0.973 ± 0.004	1.529 ± 0.092
TQ	S1 with R5	0.999 ± 0	2.699 ± 0.633
	S1 with R6	$0.999 \pm 0\ G_r$	2.684 ± 0.660

Table 4: No Previous Knowledge - UCB and TQ.

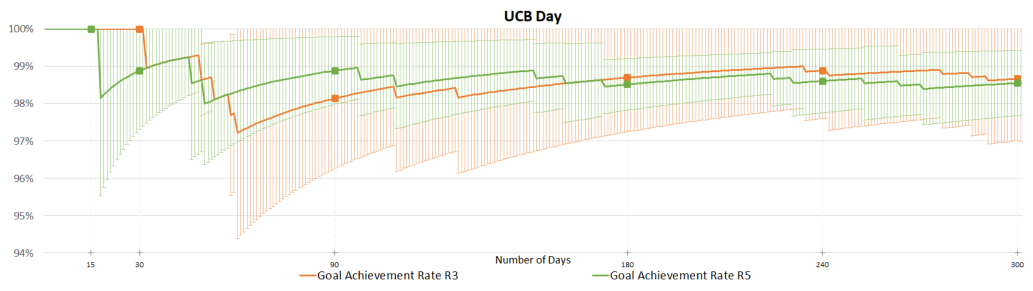
5.4. Overall Results

Best Combination - Amongst the obtained results, and following the previously defined guidelines to analyze them, the best performing combination is shown in ComboA - Figure 1. It implements UCB Day, for which a state representation is not required, with reward R5. As already detailed in section 3.1, UCB Day applies an upper confidence bound algorithm for each hour of the day, helping the model learn what action is better for each decision moment. The initialization method applied in this case was the Previously Trained Adaptive, providing a high G_r rate from the model's deployment while still allowing it to adapt over time to the user's specific routine, consequently achieving a lower N_v as time goes on. This same combo but with reward R3 provided analogous results. In contrast, R6 produced a worse user experience due to the higher penalization value for unanswered notifications, leading to a lower N_v but also a lower G_r.

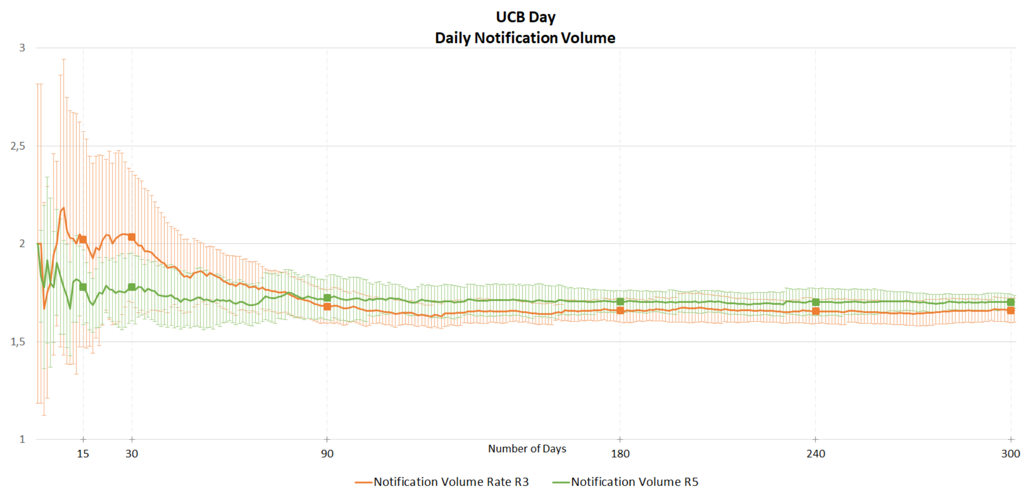
Similar results, visible in ComboB - Figure 2, were achieved in the best combination of the No Previous Knowledge method, also resorting to UCB Day and reward R5. However, ComboB's initialization method implies starting with no previous knowledge of generic user preferences, leading to initially lower G_r and slightly higher early N_v values. It takes approximately two months to achieve G_r values equivalent to the ones obtained with the initially discussed combination. Furthermore, and contrarily to ComboA, in ComboB, the reward that offered a comparable outcome utilized R6, demonstrating that, when starting with no former experience, punishing the agent more severely for sending notifications that go unanswered generates a superior performance.

It is relevant to note that although not providing the best performance, the combination that offered acceptable results consistently throughout all initialization options was implemented with TQ, state S1, and rewards R5 and R6. This affirmation is supported by Table 1. Additionally, DQN presented the most unstable and inconsistent effects, which is also reflected in Table 1, providing an overall less pleasant user experience.

Best State Representation - There was not one state which commonly displayed a better performance. This outcome was expected since the success of a state representation is highly dependent on the utilized algorithm and respective computational requirements. Yet, it could be said that S1 and S3 tend to perform better throughout all com-

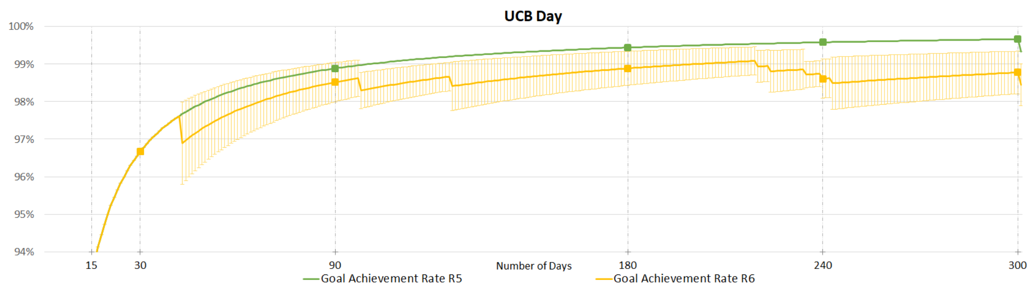


(a) Goal Achievement Rate

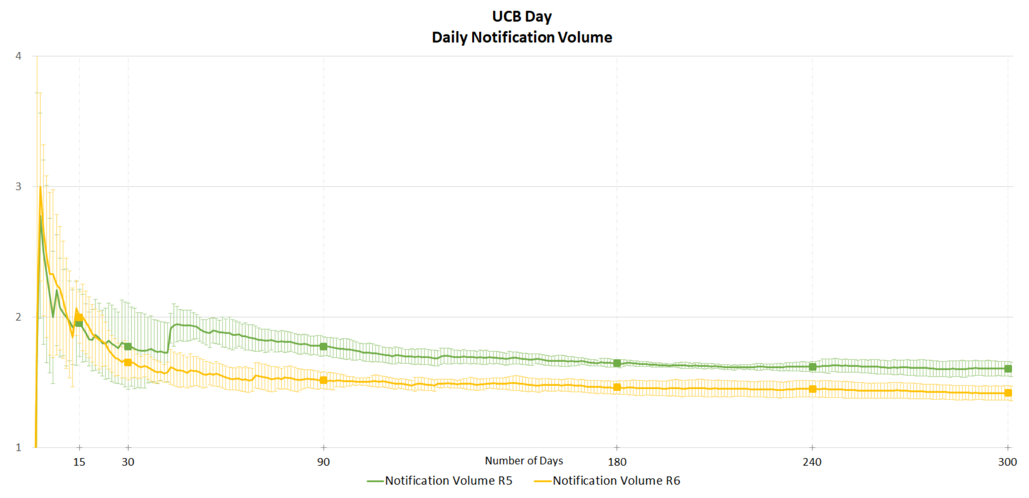


(b) Notification Volume

Figure 1: ComboA: Previously Trained Adaptive, UCB Day - average result among tested users over 300 days of training.



(a) Goal Achievement Rate



(b) Notification Volume

Figure 2: ComboB: No Previous Knowledge, UCB Day - average result among tested users over 300 days of training.

binations, revealing that more complex, detailed states are not necessarily always more efficient. Thus implying that no conclusion can yet be taken on whether patterns of opportune timings are better learned for an average day (S3) or an average week (S1). What can be confirmed is that, if appropriately designed, more straightforward states, such as S3, can contain all of the essential information, allowing for an efficient application of basic algorithms and providing superior results due to the consequent lower memory, time, and computational requisites.

Best Reward Structure - Contrarily to the state, the reward structure is highly linked and dependent on the problem and goal at hand instead of the applied algorithm. For this reason, it is possible to verify that the reward which generated a better overall performance in the tackled problem was R5. These values provided the best balance between achieving one daily answered notification and not bothering the user.

6. Conclusions & Future Work

This work aimed to build an intelligent notification system that could adequately manage interruptions, created in a mHealth self-management application, by learning what moments were the most opportune for each user throughout their day. Theoretical background and related work were analyzed, and a possible approach was developed. For the proposed methodology, a set of RL algorithms and problem formulations were reviewed, selected, and tested, in an attempt to determine the most desirable combination of initialization method, algorithm, state, and reward definition for the problem at hand. This work demonstrates that a balanced and efficient intelligent notification system can be built for the purpose of being applied to a mHealth application without requiring access to any private user information or device sensor. Hence, addressing some of the main concerns commonly shared among users, such as privacy or device and sensor-related limitations.

Possible future work could be to include more detailed user reactions which are not forced to be instant but could arrive within a predefined interval. These responses could be further elaborated by, for example, introducing oblivious dismissal (notification goes unnoticed) and intentional dismissal (people decide not to address it). Lastly, this study applies a simulator, requiring further testing in mobile devices utilized by real users with different lifestyles, diseases, contexts, and demographics. These would permit understanding which model is better applied to which type of user.

References

[1] V. P. Cornet and R. J. Holden. Systematic review of smartphone-based passive sensing for health and wellbeing. *Journal of Biomedical Informatics*, 77:120–132, 2018.

[2] M. Fiordelli, N. Diviani, and P. J. Schulz. Mapping mhealth research: A decade of evolution. *JMIR*, 15(5):1–14, 2013.

[3] Y. Fukazawa, N. Yamamoto, T. Hamatani, K. Ochiai, A. Uchiyama, and K. Ohta. Smartphone-based mental state estimation: A survey from a machine learning perspective. *JIP*, 28:16–30, 2020.

[4] Geneva: WHO. Noncommunicable diseases progress monitor 2020, 2020.

[5] B. J. Ho, B. Balaji, M. Koseoglu, and M. Srivastava. Nurture: Notifying users at the right time using reinforcement learning. *UBICOMP*, pages 1194–1201, 2018.

[6] L.-j. Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, 1993.

[7] A. Mehrotra and M. Musolesi. Intelligent Notification Systems: A Survey of the State of Art and Research Challenges. 1(1):1–26, 2017.

[8] A. Mehrotra, M. Musolesi, R. Hendley, and V. Pejovic. Designing content-driven intelligent notification mechanisms for mobile applications. *UBICOMP*, pages 813–824, 2015.

[9] A. Mehrotra, V. Pejovic, J. Vermeulen, and R. Hendley. My phone and me: Understanding people's receptivity to mobile notifications. *CHI*, pages 1021–1032, 2016.

[10] V. Mnih, K. Kavukcuoglu, and Silver. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[11] T. Rachad and A. Idri. Intelligent Mobile Applications: A Systematic Mapping Study. *Mobile Information Systems*, 2020, 2020.

[12] A. G. B. Richard S. Sutton. *Reinforcement learning : an introduction*. MIT Press, 2º edition, 2018.

[13] Y. Vaizman, K. Ellis, and G. Lanckriet. Recognizing Detailed Human Context in the Wild from Smartphones and Smartwatches. *IEEE Pervasive Computing*, 16(4):62–74, 2017.

[14] S. Vishwanath, K. Vaidya, and R. Nawal. Touching lives through mobile health-Assessment of the global market opportunity. *PwC*, 2012.

[15] P. Watkins,. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[16] WHO Global Observatory for eHealth. mHealth: new horizons for health through mobile technologies: second global survey on eHealth, 2011.

Appendix A - ExtraSensory Dataset Labels

Number	**Label**	**Label Type**	$P(A \mid L)$
1	OR_indoors	Location	0.5
2	LOC_home	Location	0.6
3	SITTING	Primary Activity	0.6
4	PHONE_ON_TABLE	Secondary Activity	0.7
5	LYING_DOWN	Primary Activity	0.6
6	SLEEPING	Secondary Activity	0.01
7	AT_SCHOOL	Location	0.5
8	COMPUTER_WORK	Secondary Activity	0.3
9	OR_standing	Primary Activity	0.5
10	TALKING	Secondary Activity	0.3
11	LOC_main_workplace	Location	0.5
12	WITH_FRIENDS	Secondary Activity	0.3
13	PHONE_IN_POCKET	Secondary Activity	0.4
14	FIX_walking	Primary Activity	0.5
15	SURFING_THE_INTERNET	Secondary Activity	0.8
16	EATING	Secondary Activity	0.8
17	PHONE_IN_HAND	Secondary Activity	0.8
18	WATCHING_TV	Secondary Activity	0.9
19	OR_outside	Location	0.5
20	PHONE_IN_BAG	Secondary Activity	0.4
21	OR_exercise	Primary Activity	0.15
22	DRIVE_-_I_M_THE_DRIVER	Secondary Activity	0.1
23	WITH_CO-WORKERS	Secondary Activity	0.1
24	IN_CLASS	Location	0.2
25	IN_A_CAR	Location	0.5
26	IN_A_MEETING	Location	0.08
27	BICYCLING	Primary Activity	0.05
28	COOKING	Secondary Activity	0.7
29	LAB_WORK	Secondary Activity	0.1
30	CLEANING	Secondary Activity	0.2
31	GROOMING	Secondary Activity	0.18
32	TOILET	Secondary Activity	0.9
33	DRIVE_-_I_M_A_PASSENGER	Secondary Activity	0.6
34	DRESSING	Secondary Activity	0.2
35	FIX_restaurant	Location	0.6
36	BATHING_-_SHOWER	Secondary Activity	0.1
37	SHOPPING	Secondary Activity	0.3
38	ON_A_BUS	Location	0.7
39	AT_A_PARTY	Location	0.3
40	DRINKING__ALCOHOL_	Secondary Activity	0.6
41	WASHING_DISHES	Secondary Activity	0.06
42	AT_THE_GYM	Location	0.4
43	FIX_running	Primary Activity	0.1
44	STROLLING	Primary Activity	0.8
45	STAIRS_-_GOING_UP	Secondary Activity	0.5
46	STAIRS_-_GOING_DOWN	Secondary Activity	0.5
47	SINGING	Secondary Activity	0.9
48	LOC_beach	Location	0.8
49	DOING_LAUNDRY	Secondary Activity	0.8
50	AT_A_BAR	Location	0.4
51	ELEVATOR	Location	0.8

Table 5: Table of labels used in the ExtraSensory Dataset [13].