A Reinforcement Learning and Synthetic Data Approach to Mobile Notification Management

Rowan Sutton
Trinity College Dublin
Dublin, Ireland
suttonr@tcd.ie

Kieran Fraser
ADAPT Centre
Trinity College Dublin
Dublin, Ireland
kieran.fraser@adaptcentre.ie

Owen Conlan
ADAPT Centre
Trinity College Dublin
Dublin, Ireland
owen.conlan@scss.tcd.ie

ABSTRACT

Mobile push-notifications are the primary mechanism for communicating new information to smartphone users, however they can also have a negative impact on user emotions, reduce work effectiveness and decrease current task performance. Through analysing state-of-the-art research on mobile Notification Management Systems, it was identified that few open-source notification data sets and, corresponding benchmarks, have been created and the majority of NMSs apply supervised learning methods. This paper investigates the use of a, freely shareable, synthetic mobile notification data set for developing and evaluating NMS performance using Reinforcement Learning. A Q-learning and Deep Q-learning agent were trained using synthetic data and an *OpenAI Gym* environment was created for evaluation. Final results illustrated that the Q-learning and Deep Q-learning agents could predict a users action toward notifications with ≈80% success when trained and evaluated upon real or synthetic data and ≈65% success when trained on synthetic and evaluated upon real notification data.

CCS CONCEPTS

• **Computing methodologies** → **Intelligent agents**; *Simulation environments*; • **Human-centered computing** → *User models*; *Ubiquitous and mobile computing design and evaluation methods.*

KEYWORDS

push-notifications, reinforcement learning, synthetic data

ACM Reference Format:
Rowan Sutton, Kieran Fraser, and Owen Conlan. 2019. A Reinforcement Learning and Synthetic Data Approach to Mobile Notification Management. In *17th International Conference on Advances in Mobile Computing Multimedia (MoMM '19), December 2–4, 2019, Munich, Germany*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3365921.3365932

1 INTRODUCTION

Notifications are one of the main methods employed by a number of applications and technologies for re-engaging with, and communicating new information to, end-users [10]. As mobile devices are inherently ubiquitous and tend to remain within reach of their owners throughout the day [19], the potential for disruption from mobile push-notifications is much higher than from other notification sources such as desktop computers [15]. For this reason, recent research [1, 3, 7, 8, 11–13, 18] has been conducted on mobile Notification Management Systems (NMS) which aim to block or delay notifications which are not seen as useful or desired, while still allowing important notifications to be delivered immediately. Currently the majority of these state-of-the-art systems are trained using real user notification data collected *in-the-wild* and implement some form of supervised learning.

A current issue hampering development of these systems is the lack of readily available mobile notification data sets collected *in-the-wild* from real users [4]. The reasons for this are due to the highly private and sensitive information contained within notifications, which often include location, message content and mobile sensory information. Previous research [4–6] has been conducted into generating synthetic notification data sets which do not contain sensitive information but provide useful features for designing and evaluating mobile NMS. One of the goals of this work was to avail of synthetic notification generation methods to evaluate other areas of machine learning for designing mobile NMS, such as Unsupervised and Reinforcement Learning (RL), without the need to curate software and conduct an *in-the-wild* data collection study. As such, this paper aims to determine the effectiveness of a mobile NMS through implementation of two RL methods, Q-Learning and Deep Q-Learning. This was achieved through the training, evaluation and comparison of RL agents over both synthetic and real *in-the-wild* mobile notification data sets procured and developed in other recent work.

This paper is structured as follows: section 2 details current state-of-the-art work in the area of push-notifications and NMSs; section 3 outlines this paper's proposed RL approach to mobile notification management and use of synthetic data; section 4 discusses various experiments executed to evaluate the proposed solution and highlights the main results and findings; and finally section 6 offers a brief summary of the paper.

2 RELATED WORK

Prior to research exploring mobile push-notifications, extensive work was conducted in the area of desktop notifications and their associated effect on users [19]. However, attention has since shifted toward mobile due to its increasing ubiquity and levels of widespread adoption resulting in a high quantity of notifications delivered to users daily. Users receive around 60 notifications per day on average and a large proportion, 20% to 50%, of these notifications

MoMM '19, December 2–4, 2019, Munich, Germany

ACM ISBN 978-1-4503-7178-0/19/12…$15.00
https://doi.org/10.1145/3365921.3365932

are ignored [16]. This is detrimental to both application developers who want their notifications to be interacted with, and users who are interrupted by notifications which are deemed, in hindsight, unimportant.

Another reason for mobile specific analysis of notifications is the potential for distraction. This has been previously investigated within a desktop notification context where it was found that these notifications increased difficulty when attempting to continue with tasks once interrupted. In a smartphone context, similar results were found whereby switching back to an original task after an interrupting phone call could take up to four times as long. However, despite their disruptive effects studies have shown that a high proportion of these disruptive mobile notifications are interacted with directly by the user.

Impact on user emotion has also been found to correlate with the reception of specific notification types. Reception of email notifications was reported to correlate with increased levels of stress and annoyance [15], however the blocking of these notifications would increase user habits of phone checking to ensure that nothing of importance was missed [14].

There are many challenges to acquiring mobile notification data such as: capturing and interpreting a large number of mobile sensory data points; capturing notification content in an unobtrusive, holistic and resource-sensitive manner; and privacy concerns surrounding the collection and storage of sensitive user notification data. Current systems for managing mobile notifications are insufficient at dealing with these issues. OS level control provides management on a per-app basis, although it does not offer an intelligent solution for managing notifications by their content or context. Do-not-disturb mode or direct reduction in delivery of notifications is also not viable as it would increase mobile phone interactions due to user FOMO (Fear Of Missing Out)[17]. App developers are also implementing their own intelligent notification delivery systems, however these are unable to utilize the full variety of data available to mobile devices without requesting application access to privacy-sensitive data from sensors and other sources. App developers can also be biased toward maximising the Click-Through-Rate (CTR) of notifications, which results in user engagement data being exploited for commercial gain as opposed to being used to curate healthier digital experiences.

A NMS was designed by Corno, et al. [1] to manage the delivery of notifications for a variety of IoT devices and aimed to avoid the same notification being read by the user on multiple devices. While this system was not designed exclusively for mobile devices, it does demonstrate many of the same design principles used when creating a mobile NMS such as decisions on notification delivery time, different methods of notification delivery, and the use of device sensors as input features for the system.

PrefMiner [12] was designed by Mehrotra, et al. to automatically create rules for notification delivery from user interaction with notifications. The system was also designed to implement its machine learning in a human interpretable manner by labelling the rules with the most common keywords of the relevant notifications. This system determines the best time for delivery but can also block low priority or low interaction notifications. In order to avoid the blocking of reminders such as calendar events which receive no user interaction, these reminder notifications are identified and separated from the rest of the data set. One possible difficulty is that it can be difficult to identify which notifications are reminder notifications for a new generic application. The remaining non-reminder notifications are clustered via unsupervised learning based on their titles and association rules are created based on features such as user response to the notification, arrival time and location. During the 15-day implementation of the system, 179 rules were suggested to the participants out of which 102 rules were accepted. One of the downsides of this system is the requirement for high levels of direct user interaction.

An off-device supervised classification system developed by Pradhan et al. [16] was designed to determine notification importance using both direct collection of notification importance to users and a more passive monitoring of user notification interaction. A variety of different supervised learning systems were used and analyzed, finding that support vector machines were the slowest implementation and decision trees were the fastest while maintaining good prediction quality. The overall precision, accuracy and recall of the systems were over 87%.

In contrast, a NMS designed by Huang and Kao [8] focused on increasing the CTR of advertising notifications, as opposed to improving user experience with mobile notifications. The system performs this by filtering out unwanted advertising notifications and by raising notifications about the smartphone condition such as phone temperature to prompt the user to open the notification drawer more often and view more advertising focused notifications. The system takes notification feature data and sends it to a remote server for processing by a deep neural network (DNN). Due to the mixture of a DNN implementation and a large feature space used, the system cannot be deployed on-device for mobile. This raises privacy concerns since highly sensitive mobile data is being uploaded to a remote server and there is difficulty in determining whether mobile data leaving a smartphone is being sent to malicious applications. This system resulted in increased app retention by around 2%-2.7%, a reduction in the number of notifications raised to the user and an increased CTR of notifications.

In summary, state-of-the-art NMSs primarily use supervised learning algorithms trained on notification data collected from users after first developing and executing *in-the-wild* data collection studies. In contrast, this paper discusses an unsupervised learning approach toward mobile notification management and the use of synthetic notification data sets for efficient initial NMS training and evaluation. Results are then compared using real data sets and conclusions are drawn regarding the utility of the method.

3 REINFORCEMENT LEARNING FOR MOBILE NOTIFICATION MANAGEMENT

The goal of a NMS is to deliver notifications to a user in the correct context so as to maximise the engagement of, and usefulness for, the user. To put this in the context of a Reinforcement Learning problem, a state is defined by the current context of the user and the features of an incoming notification. An action is defined as the user's engagement with the delivered notification, *open* or *dismiss*. As future notifications to be delivered cannot be determined based on a current action and state, a model-free algorithm is required.

Three different model-free RL algorithms were considered for implementing a NMS. These were Monte Carlo, Temporal Difference (TD) and Q-learning. Of these three, Q-learning was chosen due to Monte Carlo's relative computational inefficiency and possibility of missing certain notifications in the data set, and TD's stronger dependence on state order which introduces an extra system parameter to monitor. An implementation of Deep Q-learning was also created for comparison.

3.1 States, Actions and Rewards

Following a Markov Decision Process model, the states for the Q-learning and DQN agents were the different possible combinations of feature values from the notification. The actions were whether the user engaged with the notification. The reward values were determined by whether the action value predicted by the RL agent matched the ground-truth action value of that notification present in the data set.

Figure 1: Visualization of the Q-Table decision process given state S1

3.2 Q-learning

Q-learning operates by creating a Q-Table where each row corresponds with a possible notification state, and each column corresponds with a possible action. As the agent is trained, Q-values, which indicate a confidence-level that an action is optimal in a given state, are represented as a floating-point value in the cell corresponding to that state's row and action's column. When the trained agent is implemented and given an input state, a look-up is performed for the corresponding row in the Q-Table, and the column with the maximum value in that row is the action to perform as shown in Figure 1.

Each Q-value of the Q-Table correspond to state-action pairs and are updated according to the 1-step Q-learning algorithm proposed by Watkins [20], illustrated in equation 1. This updates the value at a given index, Q_{t+1}, for the Q-Table using the previous value at that index, Q_t, the reward, R_{t+1}, for taking action a in state s and the maximum Q-Table value for the next state out of all possible actions for that state. A learning rate, α, of 0.7 and a discount rate, γ, of 0.618 were chosen for this study as it was found they produced good results, however both values are open to further experimentation - hyperparameter tuning will be a focus of future work.

Table 1: Notification and context-relevant features present in data set.

Feature	Explanation
app	The mobile application which created the notification
category	The category assigned to the notification by the notification/app creator
subject	The subject inferred using Google's *Content Classification* API
priority	The priority assigned to the notification by the notification/app creator
number-of-updates	The number of times the notification was updated to relay new information
contact significant	Whether the contact was significant relative to the given context
time-of-day	The time at which the notification was created
day-of-week	The day on which the notification was created

$$Q_{t+1} \leftarrow Q_t + \alpha \left[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q_t \right) \right] \quad (1)$$

3.3 Deep Q-learning

As Q-Tables expand with the size of the state space, they can become quite large and inefficient depending on the number of features and actions inherent to the problem. A Deep Q-learning Network (DQN) agent was subsequently applied to the problem as a comparison to Q-learning. The DQN replaces the Q-Table in Q-learning with a Deep Neural Network (DNN). The DNN accepts states as input, instead of the state-action pairs used for Q-learning, and optimizes its weights based on the Huber Loss [9] function as shown in equation 2. A clip-delta of 1 was used in this DQN implementation.

$$L(y, f(x)) = \begin{cases} [y - f(x)]^2 & \text{for } |y - f(x)| \leq \delta, \\ 2\delta\,|y - f(x)| - \delta^2 & \text{otherwise.} \end{cases} \quad (2)$$

3.4 Data

The notification data used for this work was collected during an *in-the-wild* study of mobile notification-engagements of 15 users [5, 6]. The *WeAreUs* app was used to capture both notification and user-context features. The original data set contains data points on over 30,000 notifications, 4,940 smartphone usage logs and 291 questionnaires.

A synthetic notification data set was subsequently created using the WeAreUs data as seed to a Generative Adversarial Network (GAN). The generator of the GAN was trained to output data matching that of the original data set, thus creating a synthetic data set which could be freely shared and used for training intelligent systems.

In this work, the synthetic data set is used to train two RL agents on the problem of notification management in order to evaluate the potential of using synthetic data in place of real data for enhancing the privacy of the user. This method also facilitates the ability to

freely share data sets which would otherwise be too sensitive to distribute, such as notifications, within the wider research community, better enabling benchmarking, reproducibility and replicability of results. An explanation of the features present in the data set can be found in Table 1. Note that not all features present were used for predicting notification engagements. Constraints due to *size of state space* will be discussed further in the following sections. The synthetic data used for experimentation in this work can be found in the Github repository of Gym-push [2].

3.5 Gym

OpenAI Gym is an open source interface to Reinforcement Learning (RL) tasks. It provides environments for researchers to benchmark RL agents on simulations of real-world problems. *Gym-push* [2] is a custom OpenAI Gym environment developed specifically for training and evaluating agents attempting to better manage notification on behalf of smartphone users. The environment simulates push-notifications arriving to a user's device, the context in which the user receives the notification and the subsequent reward received for engagements made by the user. *Notif*, created for this work, has been added as an environment of Gym-push and is freely available to the research community for training RL agents to manage notifications.

4 EXPERIMENTATION

In order to gauge the ability of the proposed RL agents applied to Mobile Notification Management, the RL agents were first trained using the synthetic data of an individual user and evaluated using a number of metrics. The same RL agents were then trained using real data from the same individual and evaluated using the same metrics, for comparable results. To further assess the utility of synthetic data, the RL agents were subsequently trained using synthetic data and evaluated on real data. Finally, the method of training and evaluating the RL agents was expanded to include multiple users to ascertain if performance results achieved on the individual user could be extrapolated.

4.1 Set-up

Both RL agents were evaluated using the OpenAI Gym environment, *Notif*, found in Gym-push. The data sets used for training, validation and evaluation purposes were loaded into the environment which would then simulate the real-world delivery of notifications paired with the transitioning contexts of the user. The data sets used were all derived from the WeAreUs study and are as follows:

(1) Individual User (Synth & Balanced) - *This data set is comprised of 6,075 synthetic notifications generated from original notification data collected from an individual user in the WeAreUs study. For training and testing purposes the data set was split into the following sample sizes and balanced (process outlined below) 50, 100, 250, 500, 1000, 2500, 5000.*
(2) Individual User (Real & Balanced) - *This data set is comprised of 6,075 original notifications collected from the same individual in the WeAreUs study. This data was balanced and split into the same number of sample sizes as above.*
(3) Individual User (Real & Unbalanced) - *This data set is identical to above, but not balanced in order to evaluate performance in a real-world context where balancing may not be possible.*
(4) Multiple Users (Real & Unbalanced) - *This data set is comprised of 1000 original notifications collected from multiple users in the WeAreUs study. For the same purposes, this data was also not balanced.*

4.1.1 Evaluation Metrics. To evaluate the performance of both RL agents, 10-fold cross validation was implemented and the following metrics were calculated given subsequent agent predictions of whether a notification would be *opened* or *dismissed* by the user: *precision*, *accuracy*, *recall* and *F1 score*. These metrics were measured for each k-step in the 10-fold cross validation evaluation and were averaged across the 10 k-steps. For evaluating the computational performance of the RL agents, the time taken for the agents to fully train and be evaluated were also measured. Information regarding the agents training process was also recorded in the change of the epsilon value and percentage training reward for each episode in the training process. The percentage training reward shows which percent of the training notifications were predicted correctly during the agents training process and is evaluated as the number of positive reward signals in that episode divided by the total number of notification states used in that episode. The epsilon value corresponds approximately to the percentage of actions taken by the agent which are chosen randomly instead of using the learned preferred action.

4.1.2 Data Preprocessing. A common trend found in studies of mobile notification interaction is that the majority of notifications are usually dismissed by the user with a small sample being opened. This results in significant class imbalance in the data. Two methods of dealing with class imbalance are oversampling or undersampling. Oversampling operates by generating extra data of the underrepresented class in order to have a class balanced data set. Undersampling reduces the number of samples of the over-represented class. As this work already relies upon synthetically generated data, oversampling would result in the data set being less representative of the true population from which it originated. Conversely, undersampling, which was the method chosen in this work, exploits the benefit of using synthetic data as additional data can be generated until a class-balanced data set of required size is achieved. The end result of this is a synthetic data set with approximately the same number of notifications opened as there are dismissed. Real data was also used in some of the experiments for comparison purposes. In those cases, undersampling was used to achieve balance unless otherwise stated. However, undersampling would not be used in practice to balance data sets since there is generally a low amount of notification data available and the RL agents would want to maximise its utilization. This is a key problem area that the synthetic data generation process can address.

4.2 Evaluating RL agents with synthetic data

In this section, the RL agents were trained and evaluated upon the *Individual User (Synth & Balanced)* data set. Unless otherwise stated, the set of notification features exploited by both agents for predicting open/dismiss actions were { app, category, time-of-day }.

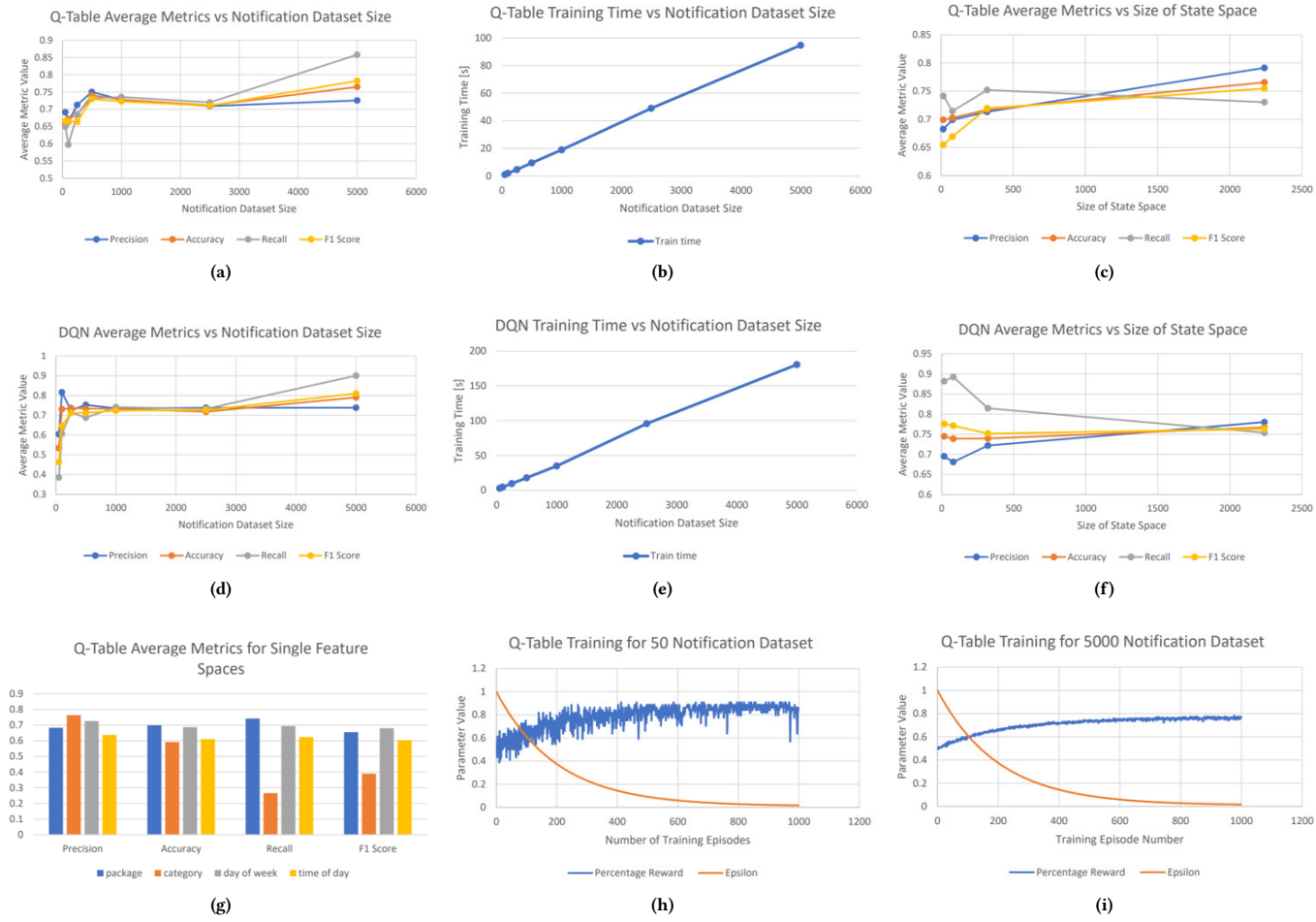

Figure 2: Evaluation of RL agent performance with synthetic notification data

4.2.1 Size of data set. The analysis of performance versus data set size is an important factor as it corresponds to the number of notifications required by the agents to start achieving a high level of classification performance. Note that the data set sizes shown are the sizes of the overall notification data set used in the 10-fold cross-validation. As a result, the training and testing set sizes are 90% and 10% of the overall data set size respectively. Figure 2a illustrates how precision, accuracy, recall and F1 score are affected by the number of notifications used by the Q-learning agent and Figure 2d illustrates the same for the DQN agent.

For data set sizes below 500 notifications, there were lower values for all metrics in both agents. In particular, the DQN agent had very low recall and F1 Score values which indicates that for low numbers of training notifications, the DQN agent was predisposed towards simply *dismissing* most notifications when implemented. The performance of both agents saturate from 500 to 2500 notifications after which an increase in the metrics of recall, F1 Score and accuracy occurs for a data set size of 5000. For 5000 notifications, the Q-learning agent displays a high accuracy of 76.5%, a very high recall value of 85.8% and high F1 score of 78.2%. The DQN agent showed comparably better results with an accuracy of 79.1%, a recall of 90.1% and an F1 score of 81.0%.

4.2.2 Time to train. The training times were measured for both agents to give an indication of how the computational performance changed with the size of the notification data set used. Figures 2b and 2e illustrate a linear relationship between the size of the notification data set and the time taken to train the Q-learning and DQN agents respectively. The training time for the DQN agent was approximately twice as long as the Q-learning agent. Testing times were also analysed for both agents with the Q-learning agent executing ≈40 times faster than the DQN agent.

4.2.3 Size of state space. To ascertain if the number of features used to define the notification states had an impact on the RL agent performance, different numbers of features were chosen for evaluation, illustrated in Table 2. An overall data set size of 1000 notifications was chosen for evaluating each of the following feature state spaces.

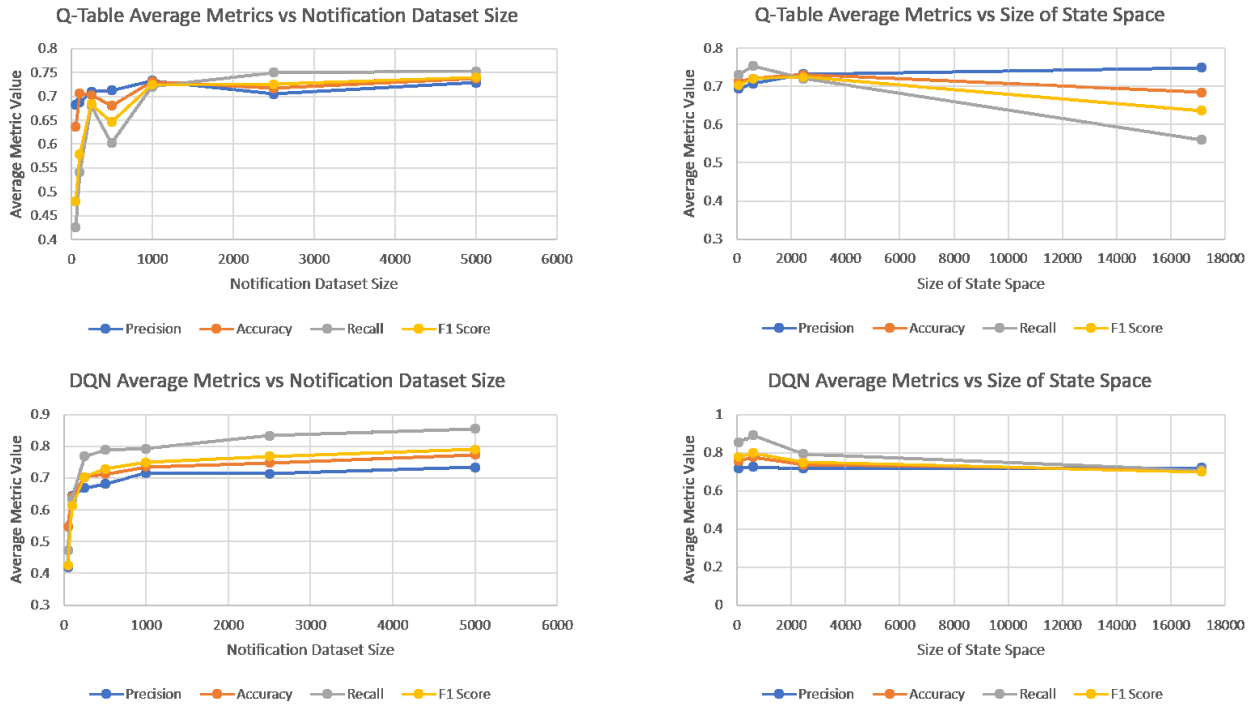

Figure 3: Evaluation of RL agent performance with real notification data

Table 2: Number of features used and the corresponding size of notification state space

# Features	Feature Types	State Space Size
4	app, category, time-of-day, day-of-week	2240
3	app, category, time-of-day	320
2	app, category	80
1	app	16

Figures 2c and 2f illustrate the precision, accuracy, recall and F1 score values across the differing state spaces for both the Q-learning and DQN agents respectively. For the Q-learning agent, there were improvements in all metrics except recall as the number of features increased. It was also found the standard deviation across k-steps also decreased as more features were used leading to more consistent performance of the agent. By contrast, for the DQN agent, the F1 score value changed very little between different state space sizes. Instead there was a convergence of metric values as the state space grew with an initially high recall above 87.5% and lower precision below 70% which converge to values between 75% to 80% as the state space increased in size. The standard deviation values for the DQN metrics were found to be very low for all state spaces.

4.2.4 Feature performance. The Q-learning agent was also analysed for performance when using just one feature to define the notification state space. This analysis took place to identify which features were more effective for defining the notification state space for increased prediction performance. Each of the four feature types {app, category, time-of-day, day-of-week} were used independently and the agent performance was measured for each. A data set size of 1000 synthetic notifications was used for these measurements.

Figure 2g illustrates that the app and day-of-week features showed the best performance with F1 scores ≈65-70%. The time-of-day feature showed slightly worse performance with metrics approximating 60%, while category resulted in an F1 score below 40%.

4.2.5 Agent training metrics. Figures 2h and 2i depict the epsilon and percentage reward values during the training process of the Q-learning agent when a small and large data set are used. These metrics were taken for each episode of the 1000 training episodes used. They were measured to give insight into the training process of an RL agent and how these values change with data set size. By comparing the 50 notification data set size results with those of the 5000 data set size, a decrease in variation of training reward between training episodes can be identified. This supports the hypothesis that by using a larger training data set, the results are more representative of the overall data set.

4.3 Evaluating RL agents with real data

In this section, the RL agents were trained and tested upon the *Individual User (Real & Balanced)* data set. Unless otherwise stated, the set of notification features exploited by both agents for predicting open/dismiss actions were { app, category, time-of-day }.

4.3.1 Size of data set. Figure 3 illustrates the precision, accuracy, recall and F1 score of both agents tasked with predicting notification actions using different real data set sizes. Compared with results of synthetic data discussed in section 4.2.1, it can be seen that the agents perform slightly worse on the real data. The F1 score values of the Q-learning agent are below 50% starting out compared to its synthetic counterpart which begins at close to 60% on the smaller data sets. The max F1 score value of the Q-learning agent, occurring at the 5000 data set size mark, is also less impressive when evaluated on real data, ≈75% compared with the synthetic value of ≈80%. A suggested reason for the higher performance in agents using synthetic data is that the synthetic data is less nuanced and diverse than the real data, hence the complexity of the classification problem is reduced. The relatively low difference between max F1 score values however, suggests that synthetic data could be a viable solution for training intelligent agents when real data is difficult to extract and exploit. The DQN agent exemplifies this point, with a max F1 score value of ≈80% using a data set size of 5000 in both real and synthetic cases.

4.3.2 Size of state space. Similar to the analysis of section 4.2.3, this section attempts to identify if the number of features used to define the notification states has an impact on the RL agent performance when using real data. The number of features and corresponding size of states are illustrated in Table 3. Note that compared with section 4.2.3, the number of state sizes has increased even though the number of features are identical. This is due to the synthetic data not fully capturing all features present in the original data set and highlights the current limitation of the synthetic generation process to fully represent the original seed data.

In contrast to agent performance using synthetic data discussed in section 4.2.3, increasing the state space size actually reduces the performance of agents when applied to real data. Figure 3 illustrates the F1 score of both agents over differing state space sizes. The Q-learning has a max value of 74% when using 3 features, but drops to 64% when using 4 features, which is lower than using just 1 feature alone. Similarly the DQN agent reaches peak performance using just 2 features before a slight drop in performance leads to saturation at 65%, again lower than using just the app feature for representing the notification state.

4.3.3 Feature performance. As with section 4.2.4, the Q-learning agent was again used for identifying feature importance in predicting a user action toward notifications. Interestingly, when applied to real data, the app feature performs best with values greater than 70% across all metrics. The category feature, which previously performed worst, is now the next best predictor of user action after the app. This again highlights the need for improving the synthetic generation process to facilitate all feature values present in the original data. The results are illustrated in Figure 5.

Table 3: Number of features used and the corresponding size of notification state space

# Features	Feature Types	State Space Size
4	app, category, time-of-day, day-of-week	17136
3	app, category, time-of-day	2448
2	app, category	612
1	app	51

4.4 Train on Synthetic, Test on Real evaluation of RL agents

In this section, the RL agents were trained upon the *Individual User (Synth & Balanced)* data set and tested upon the Individual User (Real & Unbalanced) data set. This method of Training on Synthetic and Testing on Real (TSTR), was used to ascertain whether synthetic notification data could be used in place of real data to predict user actions with respect to incoming notifications. The set of notification features exploited by both agents for predicting open/dismiss actions were { app, category, time-of-day }.

4.4.1 Size of data set. Similar to sections 4.2.1 and 4.3.1, both agents were trained and evaluated using different sized notification data sets. However, in contrast to previous sections, the test data remained fixed at 1000 notifications and was made up of real notification data. The agents were trained using the synthetic data sets of varying sizes and, because of this, the models, during testing, were exposed to states which were not present during training. Future work will explore how entity embedding may aid in mapping unseen states to similar, trained states, but for the purposes of this work and evaluating performance of the RL agents, a random action was chosen when unknown states appeared. The results are illustrated in Figure 4. As expected, the performance of both the Q-learning and DQN agents drop below previous benchmarks set in Figures 2 and 3. The initial F1 score of the Q-learning agent fell just above 20% and achieved a max value of 61% when trained upon 5000 notifications. The DQN agent had an F1 score below 20% when trained with 50 notifications and a max value of 63% when trained on 5000. Both achieved performance levels well above a random benchmark of 50% which suggests that the agents trained on synthetic data could be applied in cases where use of real data is not possible.

4.4.2 Unknown state spaces. The limitations of performance can also be attributed to the number of unknown cases forcing the RL agents to take a random action. Figure 4 also illustrates how, when the number of unknown states is reduced, the performance of the agent increases and becomes more stable. This occurs with greater amounts of synthetic data generated and made available for training. Improving the synthetic generation process such that all unique feature values get represented in the synthetic training data would also aid in minimizing the number of unknown states encountered during testing/deployment in-the-wild.

Figure 4: RL agent performance when trained using synthetic notification data and evaluated on real notification data (TSTR)

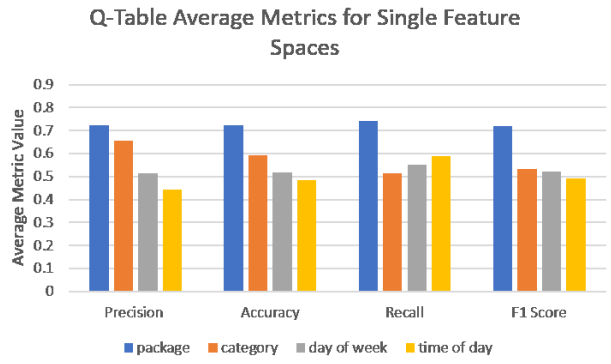

Figure 5: Analysis of feature importance applied to real notification data

4.5 Evaluating RL agents across multiple users

In this section, the RL agents were trained and evaluated upon the *Multiple Users (Real & Unbalanced)* data set. The set of notification features employed by both agents for predicting open/dismiss actions were { app, category, time-of-day }.

The results, illustrated in Figure 6, highlight that for different user personas, performance varies when attempting to predict notification engagement on behalf of the user. As expected, based on previous results discussed, the DQN agent achieves the maximal F1 score of 76% and improved F1 score values across users 1, 2 and 5 over the Q-learning agent. However, the Q-learning agent shows higher consistency over all users, highlighted by its improved performance over the DQN agent for users 4 and 6 and its comparable results across all other users. The size of the training data set, 1000 notifications, may be a contributing factor toward the DQN's limited performance for users 4 and 6 as, previously discussed, this agent performs best on larger data sets while the Q-learning agent performs well even on limited data.

The user's CTR is superimposed over the performance of each agent on each user. As can be seen across users 1 to 3, the performance of the RL agents is consistent across differing user engagements. User 3 tends to accept up to 60% of notifications while user 2 tends to dismiss up to 80%. This shows how the RL agents are successful at learning both types of engagement levels on behalf of the users.

Through comparison of user 2, who has a low CTR but high F1 score over each agent, and users 4 and 6, who also have low CTR's but much lower F1 scores over each agent, it was identified that the feature distributions for the app, time-of-day and category were quite different between these users. 90% of notifications for user 2, for example, had an *unknown* category, while users' 4 and 6 had a much more diverse range of notification categories made up of *email*, *msg*, *alarm* and *calls*. Similarly, users' 4 and 6 received most notifications through *messaging* and *email* apps in contrast to user 2 who received mainly *weather* and *alarm* notifications. The diversity of feature values found in users' 4 and 6 may be an indication of

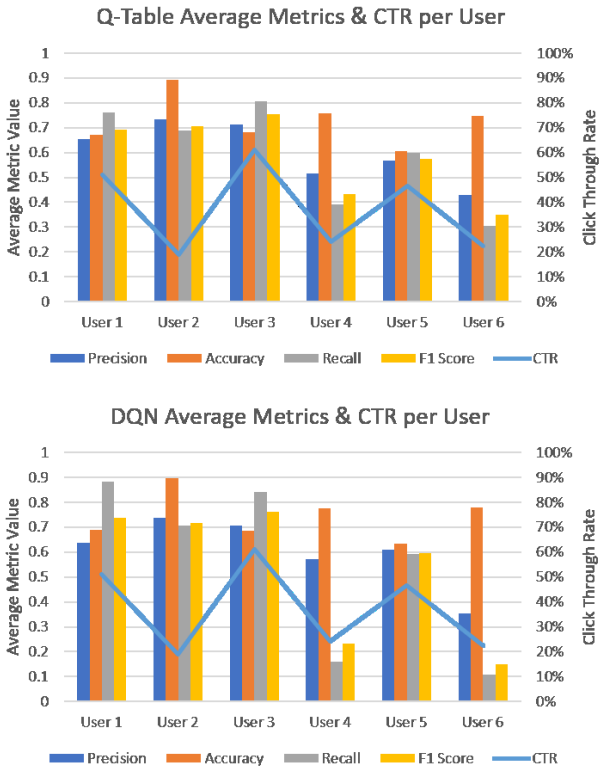

Figure 6: Evaluation of RL agent performance with multiple real user notification data

why both agents performed worse for these users, due simply to the complicated range of notification-engagement behaviours they exhibit over all feature values. However, as there may be a number of confounding variables at play, identifying if causation is present is left to future work.

In summary, it can be seen that both agents have the ability to perform well over a number of different user notification-engagement types, with the Q-learning agent being preferred in scenarios of limited data and the DQN agent when larger data sets are at its disposal.

5 LIMITATIONS & FUTURE WORK

The work discussed in this paper has a number of associated limitations, including: the small set of users present in the WeAreUs data set; the restricted set of notification/context features used; and the fixed hyperparameters of the models. The small set of users in the WeAreUs study can be attributed to unwillingness of potential participants to have their notifications monitored due to the the sensitive nature of notification content. Further work will need to address if and how these concerns can be alleviated to facilitate the gathering and use of notification and smartphone data for the purposes of improving push-notification experiences. The use of generative modeling techniques to create synthetic data is a potential solution, as discussed in this paper. Additionally, while supplementing the number of features made available to both agents would increase the state-space and subsequent computing power necessary for training, it may also positively impact performance metrics, as might optimizing the hyperparameters of both models. Therefore, both tasks are recommended for future evaluation. As the resources used in these experiments can be found in the Gym-push repository [2], the research community is invited to work with us in improving the performance of both agents to facilitate smarter push-notifications.

6 CONCLUSION

The goal of this work was to design, implement and evaluate a new method for managing notifications on behalf of users, without explicit use of their privacy-sensitive data. The solution proposed by this paper was a Reinforcement Learning approach whereby two agents were trained upon a synthetic data set generated from a previous in-the-wild study of mobile push-notifications. The results from the Q-learning and Deep Q-learning agents were evaluated using a number of metrics and a custom *OpenAI Gym* environment.

Findings illustrated that the Reinforcement Learning methods chosen could predict user engagements toward notifications up to ≈80% of the time when trained and tested upon real and synthetic data respectively, and up to ≈65% when trained upon synthetic and evaluated upon real notification-engagement data. These results compare favourably with current state-of-the-art approaches.

ACKNOWLEDGMENTS

This research is supported by the ADAPT Centre for Digital Content Technology under the SFI Research Centres Program (Grant 13/RC/2106) and is co-funded under the European Regional Development Fund.

REFERENCES

[1] Fulvio Corno, Luigi De Russis, and Teodoro Montanaro. 2015. A context and user aware smart notification system. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. IEEE, 645–651.
[2] Kieran Fraser and Rowan Sutton. 2019. Gym-push. Retrieved October 20, 2019 from https://github.com/kieranfraser/gym-push/tree/master/MoMM2019
[3] K. Fraser, B. Yousuf, and O. Conlan. 2016. A context-aware, info-bead and fuzzy inference approach to notification management. In *2016 IEEE 7th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*. 1–7. https://doi.org/10.1109/UEMCON.2016.7777832
[4] Kieran Fraser, Bilal Yousuf, and Owen Conlan. 2017. Synthesis & Evaluation of a Mobile Notification Dataset. In *Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization*. ACM, 179–184.
[5] Kieran Fraser, Bilal Yousuf, and Owen Conlan. 2019. Generation and Evaluation of Personalised Push-Notifications. In *Adjunct Publication of the 27th Conference on User Modeling, Adaptation and Personalization (UMAP'19 Adjunct)*. ACM, New York, NY, USA, 223–224. https://doi.org/10.1145/3314183.3323683
[6] Kieran Fraser, Bilal Yousuf, and Owen Conlan. 2019. Scrutable and Persuasive Push-Notifications. In *International Conference on Persuasive Technology*. Springer, 67–73.
[7] Joyce Ho and Stephen S. Intille. 2005. Using Context-aware Computing to Reduce the Perceived Burden of Interruptions from Mobile Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '05)*. ACM, New York, NY, USA, 909–918. https://doi.org/10.1145/1054972.1055100
[8] TonTon Hsien-De Huang and Hung-Yu Kao. [n. d.]. C-3PO: Click-sequence-aware deeP neural network (DNN)-based Pop-uPs recOmmendation. *Soft Computing* ([n. d.]), 1–7.
[9] Peter J Huber. 1992. Robust estimation of a location parameter. In *Breakthroughs in statistics*. Springer, 492–518.
[10] Shamsi T Iqbal and Brian P Bailey. 2010. Oasis: A framework for linking notification delivery to the perceptual structure of goal-directed tasks. *ACM Transactions on Computer-Human Interaction (TOCHI)* 17, 4 (2010), 15.
[11] Nicky Kern and Bernt Schiele. [n. d.]. Context-aware notification for wearable computing. In *Proceedings of the 7th IEEE International Symposium on Wearable*

Computers (ISWC'03). 223–230.

[12] Abhinav Mehrotra, Robert Hendley, and Mirco Musolesi. 2017. Interpretable machine learning for mobile notification management: An overview of prefminer. *GetMobile: Mobile Computing and Communications* 21, 2 (2017), 35–38.

[13] Tadashi Okoshi, Julian Ramos, Hiroki Nozaki, Jin Nakazawa, Anind K. Dey, and Hideyuki Tokuda. 2015. Reducing Users' Perceived Mental Effort Due to Interruptive Notifications in Multi-device Mobile Environments. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '15)*. ACM, New York, NY, USA, 475–486. https://doi.org/10.1145/2750858.2807517

[14] Antti Oulasvirta, Tye Rattenbury, Lingyi Ma, and Eeva Raita. 2012. Habits make smartphone use more pervasive. *Personal and Ubiquitous Computing* 16, 1 (2012), 105–114.

[15] Martin Pielot, Karen Church, and Rodrigo De Oliveira. 2014. An in-situ study of mobile phone notifications. In *Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services*. ACM, 233–242.

[16] Swadhin Pradhan, Lili Qiu, Abhinav Parate, and Kyu-Han Kim. 2017. Understanding and managing notifications. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 1–9.

[17] Andrew K Przybylski, Kou Murayama, Cody R DeHaan, and Valerie Gladwell. 2013. Motivational, emotional, and behavioral correlates of fear of missing out. *Computers in Human Behavior* 29, 4 (2013), 1841–1848.

[18] Yujue Qin, Tanusri Bhattacharya, Lars Kulik, and James Bailey. 2014. A Context-aware Do-not-disturb Service for Mobile Devices. In *Proceedings of the 13th International Conference on Mobile and Ubiquitous Multimedia (MUM '14)*. ACM, New York, NY, USA, 236–239. https://doi.org/10.1145/2677972.2678003

[19] Alireza Sahami Shirazi, Niels Henze, Tilman Dingler, Martin Pielot, Dominik Weber, and Albrecht Schmidt. 2014. Large-scale assessment of mobile notifications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 3055–3064.

[20] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.