Evaluating Synthetic Notification Trained Reinforcement Learning for Mobile Notification Management Systems

Rowan Sutton

MAI in Computer Engineering

Trinity College Dublin

Supervisor: Dr. Owen Conlan

Submitted to the University of Dublin, Trinity College, April, 2019

I, Rowan Sutton, declare that the following dissertation, except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other University; and that the library may lend or copy it or any part thereof on request.

Signature:

Date:

1 Summary

Initially this report analyses the importance of mobile notifications for communicating information to users and how the ubiquity of mobile devices and increase in the number of incoming mobile notifications created cause for research into mobile specific notifications. This research highlighted how mobile notifications can have a negative impact on user emotions, reduce work effectiveness and decrease current task performance. Complexities in the reduction of notifications were also investigated such as the importance of fast messaging and high priority notification delivery despite their possible negative emotional impact, the management of "reminder notifications" which are not interacted with by users but still impart useful information, and the technical issues that can be caused by blocking or interrupting notifications which are required for app operation.

For these reasons, mobile notification management systems (NMSs) were developed. The problems with currently implemented solutions such as OS level control, do not disturb mode and app specific importance learning were discussed, which lead to analysis what state-of-the-art NMSs are currently being researched.

Discussion of the current state-of-the-art systems developed by (Corno et al., 2015), (Mehrotra et al., 2017), (Pradhan et al., 2017), and (Huang and Kao, 2019) found gaps in research for both the areas of reinforcement learning in NMSs and the use of fully synthetic notification datasets for training and evaluation of NMS. The benefits of using a synthetic notification dataset include an increased level of privacy sensitivity when compared to other "in-the-wild" datasets and the ability to scale the size of the dataset for the use case.

For this reason both a Q-Learning and Deep Q-Learning system were developed using a synthetic notification dataset created by (Fraser, 2018). These systems were implemented through Python scripts on a non-mobile computer to simplify their implementation and allow focus on a larger variety evaluation parameters and metrics. An OpenAI Gym (OpenAI, 2019) environment was created to simulate the mobile device raising notifications to the machine learning systems.

These systems were evaluated using 10-fold cross-validation for the metrics of precision, accuracy, recall, F1 Score and computation time. The training reward over time was also measured for each k-step of the 10-fold cross validation.

The parameters changed for evaluation were the size of the notification dataset and the number of features used to define the notification state space. Analysis of performance for different single feature spaces was also performed for the Q-Learning implementation.

The results found that the Q-Learning and Deep Q-Learning systems obtained a maximum performance in machine learning metrics of precision, accuracy, recall and F1 Score of approximately 80%. The results also indicate that a Q-Table implementation should be used for small to medium datasets, a computation time requirement or the need for real-time implementation, whereas the DQN implementation should be used for large datasets, GPU optimized systems or large feature spaces particularly if there are memory constraints.

Overall the project found an effective privacy conscious methodology for the training and implementation of mobile NMSs through the use of synthetic notification datasets.

Acknowledgements

I would like to thank my project supervisor Dr. Owen Conlan for his guidance and support throughout this research.

I would like to thank Kieran Fraser for providing the synthetic notification dataset and for his advice during the project.

TABLE OF CONTENTS

1 Summary 3

2 Introduction 10

3 Literature Review 11

3.1 Mobile Notifications 11

3.1.1 Purpose of Notifications 11

3.1.2 Studies on Desktop Notifications 11

3.1.3 Quantity of Received Notifications 12

3.1.4 Distractive Effects of Notifications 12

3.1.5 Impact of Notifications on User Emotions 13

3.1.6 Challenges in Acquiring Notification Data and Measuring User Interactions with Notifications 14

3.1.7 Notification Delivery Time 16

3.1.8 Difficulties with Reducing the Number of Incoming Notifications 17

3.1.9 Business and User Incentives for NMSs 17

3.1.10 Conclusions About Notifications 18

3.2 Mobile Notification Management Systems 19

3.2.1 Problems with User Management of Notifications 19

3.2.2 Currently Implemented Systems for Managing Notifications on Mobile 19

3.3 State of the Art Systems 20

3.3.1 A Context and User Aware Smart Notification System (2015) 20

3.3.2 Interpretable Machine Learning for Mobile Notification Management (2017) . 21

3.3.3 Understanding and Managing Notifications 2017 23

3.3.4 C-3PO: Click-sequence-aware DeeP Neural Network (DNN)-based Pop-uPs RecOmmendation 25

3.4 Comparison of SoA Systems 27

3.5 Privacy and Ethics Considerations 28

3.6 Synthetic Notification Datasets 28
3.6.1 Utility and Benefits of a Synthetic Notification Dataset 28
3.6.2 How “In-the-Wild” Data was Collected 29
3.6.3 Data Generation 29
3.6.4 Evaluation of this Dataset 30
3.6.5 Possible Downsides of Using a Synthetic Dataset 30
3.6.6 Conclusion 30
4 Methodology 31
4.1 Machine Learning Systems 31
4.2 Reinforcement Learning Algorithms 31
4.2.1 General Design 31
4.2.2 Model-Based vs Model-Free 32
4.3 Types of Reinforcement Learning Algorithms 32
4.3.1 Monte Carlo 32
4.3.2 Temporal Difference and Q-Learning 33
4.4 Q-Learning Implementation 33
4.4.1 State, Action and Reward 33
4.4.2 Exploration vs Exploitation 34
4.4.3 Q-Tables 34
4.4.4 Testing Implementation 36
4.5 Deep Q-Network Implementation 36
4.5.1 Difficulties with Q-Tables 36
4.5.2 Huber Loss Equation 37
4.5.3 Replay cache 37
4.6 Evaluation Methodology 38
4.6.1 K-Fold Cross-Validation 38
4.6.2 Metrics Used 39
4.7 Data Class Imbalance 40

5 Findings 41
5.1 Size of Dataset Used 41
5.1.1 Machine Learning Evaluation Metrics 41
5.1.2 Computational Performance Metrics 43
5.1.3 System Training Metrics 46
5.1.4 Comparison of Systems 49
5.2 Number of Notification Features 50
5.2.1 Machine Learning Evaluation Metrics 50
5.2.2 Computational Performance Metrics 53
5.2.3 System Training Metrics 55
5.3 Single Feature Analysis 56
5.3.1 Machine Learning Evaluation Metrics 57
5.3.2 Computational Performance Metrics 58
5.3.3 System Training Metrics 58
6 Analysis and Discussion 59
6.1 System Parameters 59
6.1.1 DQN Performance Issues 59
6.1.2 DQN Epsilon Rate and Dataset Size 59
6.1.3 System Learning Parameters 59
6.2 State and Action Spaces 60
6.2.1 State Space 60
6.2.2 Action Space 60
6.3 Comparison to State of the Art 60
6.3.1 System Analysis 60
6.3.2 Feature Analysis and User Interaction 61
6.4 Consequences of Using a Synthetic Notification Dataset 62
6.5 Computational Performance and Mobile Deployment 63
6.6 Change in User Context 63

6.7 System Comparisons 64
7 Conclusion 65
7.1 Overview 65
7.2 Main Findings 65
7.3 Weaknesses and Limitations 65
7.4 Recommendations 66
7.5 Further Research 66
8 References 68

2 INTRODUCTION

Mobile notifications are the main method for communicating incoming information to the user for a variety of applications. Since mobile devices are present with users throughout their day in a variety of environments, the potential for disruption from mobile notifications is much higher than from other notification sources such as desktop computers.

For this reason, research has been conducted into the area of mobile notification management systems (NMS) which aim to block or delay notifications which are not seen as useful or desired, while 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 for the development of these systems is the lack of available mobile notification datasets taken “in-the-wild” from real users. The reasons for this are due to the highly privacy sensitive information contained in these notifications which often includes location, message content and mobile sensor information. For these reasons, research has been conducted into generating synthetic notification datasets which do not contain privacy sensitive information but provide useful features for designing mobile NMS systems.

There is also scope for research into other areas of machine learning for designing mobile NMSs such as unsupervised and reinforcement learning. This paper aims to determine the effectiveness of a mobile NMS which implements reinforcement learning and is trained and evaluated using a synthetic notification dataset. By doing this the performance of using reinforcement learning for mobile NMSs is to be evaluated as well as the creation and implementation of a privacy conscious methodology for evaluating these systems.

3 LITERATURE REVIEW

3.1 MOBILE NOTIFICATIONS

3.1.1 Purpose of Notifications

Mobile notifications play an important role in how smartphone users interact with their phones. A statement by (Iqbal and Bailey, 2010) defines a "notification to refer to a visual cue, auditory signal, or haptic alert generated by an application or service that relays information to a user outside her current focus of attention." (Iqbal and Bailey, 2010). For mobile applications, they can take a wide variety of forms, from ringtones to flashing lights from the phone's L.E.D. These notifications are used to "steer the user's attention towards the newly-arrived information" (Mehrotra et al., 2016) and this information can take a variety of forms from "the arrival of a message, a new comment on one of their social network posts, or the availability of an application update" (Shirazi et al., 2014). Since notifications are the primary mechanism for delivering updates and new information to the user, it is important that they are delivered effectively.

3.1.2 Studies on Desktop Notifications

Prior to research of notifications for mobile applications, there were extensive previous studies conducted into the area of desktop computer notifications and their effect on users. "The effect of notifications created by applications in the desktop workplace, such as email clients or instant messengers applications, has been studied thoroughly in previous work [4, 5, 13, 16]" (Shirazi et al., 2014). While there are many similarities between how users interact with desktop and mobile notifications, there are differences since mobile notifications "(1) are delivered to a highly unified mechanism, (2) inform about a much larger variety of events, ranging from messages to system events, and (3) became pervasive due to the omnipresent nature of current smartphones that are virtually always with the user [12, 21, 19]."(Shirazi et al., 2014).

The pervasiveness of mobile devices throughout a users' daily life is significant as this increases the ability for users to be disrupted by notifications. This idea is reinforced by (Pielot et al., 2014) who mention how "Since we carry our phone with us throughout the day, mobile notifications continually cross the boundaries of work and private life and as such have the potential to interrupt us in a wider range of situations and contexts." (Pielot et al., 2014).

Since mobile notifications differ from conventional desktop computer notifications in these ways, there was a demand for research into notifications specifically within a mobile context.

3.1.3 Quantity of Received Notifications

A significant motivation for why the study of mobile specific notifications is an important area of analysis is the large number of mobile notifications which users receive daily. Studies by both (Pielot et al., 2014) and (Pradhan et al., 2017) have shown that users receive a high average number of notifications on a daily basis with these studies finding that users receive “63.5 notifications on average per day, mostly from messengers and email.” (Pielot et al., 2014) and “an average user receives at least 60 notifications per day” (Pradhan et al., 2017).

In addition, the study by (Pradhan et al., 2017) found “that 20% to 50% of the notifications generally get ignored by the users”(Pradhan et al., 2017) due to the high number of notifications received. Ignored user notifications are a detriment to both the app developer and the user. App developers do not want their application’s notifications to go unnoticed and mobile users do not want to miss notifications with important information.

3.1.4 Distractive Effects of Notifications

Another issue with poorly managed notifications is their potential for interruption and disruption of the user in their day-to-day work.

Within the desktop notification context, a study by (Bailey et al., 2000) which analysed the effect of interruptions on task completion found that interruptions from outside sources to the work and tasks being carried out by individuals causes an increase in feelings of anxiety and annoyance for the individual and can cause “difficulty switching back to the previously suspended primary task” (Bailey et al., 2000).

Within a smartphone context, similar results were found by (Leiva et al., 2012) where the duration of time spent on different phone app tasks was measured and compared to the duration spent on app tasks after being interrupted by a phone call. The study measured app usage from 3,611 users and found that “Phone call interruptions add a significantly high overhead on the interrupted application in comparison to those of app-switching” (Leiva et al., 2012) and that after interruption, the interrupted application’s “runtime could be increased by up to four times” (Leiva et al., 2012).

(Mehrotra et al., 2016) expands on the various impacts of interruptive notifications by stating that “previous studies have found that interruptions at inopportune moments can adversely affect task completion time [11, 12, 25], lead to high task error rate [8] and impact the emotional and affective state of the user [5, 7]. Also, users might get annoyed when they receive notifications presenting information that is not useful or relevant to them in the current context [13]" (Mehrotra et al., 2016)

In terms of notification frequency, (Pradhan et al., 2017) found that when "not managed properly, these notifications have the potential to disrupt an average user at least 4 times per hour of the productive part of their day." (Pradhan et al., 2017).

Given the frequency and high level of disruption that mobile interruptions have, poorly managed notifications can have a negative impact on work effectiveness in professional settings and on general task completion.

Even though mobile notifications cause such distraction, the study by (Pielot et al., 2014) found that "notifications were typically viewed within minutes. Social pressure in personal communication was amongst the main reasons given." (Pielot et al., 2014)

In addition, the study by (Mehrotra et al., 2016) found that while some mobile notifications caused disruption "54% of these disruptive notifications were accepted (clicked) by the users, regardless of the fact that they caused disruption" (Mehrotra et al., 2016). The explanations given by the candidates were primarily related to the notifications being of high importance.

Given the negative effect of disruptive notifications and the influence of social factors and level of importance on notification acceptance, a system which delays or blocks notifications which appear to be disruptive must also consider the level of importance the notifications have to the user. These important and social notifications must be delivered as usual without any delay or blocking by an NMS (notification management system).

3.1.5 Impact of Notifications on User Emotions

The reception of mobile notifications can have a variety of effects on the user's emotions depending on different notification features. These emotional impacts can be both negative and positive depending on the context.

In the study by (Pielot et al., 2014), "The amount of emails received during a day was correlated with increased self-reports of negative emotions. Both, subjective and objective email count, lead to higher feelings of being stressed (…), interrupted (…), and annoyed (…)" (Pielot et al., 2014).

"In contrast we found that receiving more messages is significantly correlated with increased feelings of being connected with others (…). The same positive correlation was found for the amount of social network updates and feeling connected to others (…)" (Pielot et al., 2014).

These findings emphasise that notification type and context have a strong impact on users' receptivity to notifications. The variance in emotional impact between receiving an email notification and an instant messaging notification can be explained by the type of content each medium generally delivers. Emails most often contain information which requires further action and is of a professional or important nature, leading to increased stress. By contrast, instant messages are most often used for social interaction which explains why their reception leads to feelings of being socially connected.

While the interruptions from email and other notifications which carry important information can cause negative emotional effects, the feeling of missing important information can have a similar negative impact.

In the study of mobile phone use by (Oulasvirta et al., 2012), the users' "Descriptions of the use of e-mail were mainly related to checking e-mails and, thus, achieving a sort of awareness that nothing important is missed, as opposed to actively writing messages to others" (Oulasvirta et al., 2012). In the study, this awareness of notification events was noted as one of the three main motivators for creating mobile use habits.

The associated negative emotional impact of missing important notifications such as important emails reinforces the previous findings on the disruptive effects of notifications, whereby notifications of importance should be delivered to the user without interruption. If they have high importance to the user, then they should be delivered immediately even if the user is otherwise occupied or the delivery time may appear disruptive.

3.1.6 Challenges in Acquiring Notification Data and Measuring User Interactions with Notifications

There are many different and subtle complexities in measuring and predicting users' interactions with notifications. In (Pradhan et al., 2017) a system was developed to monitor the user's interactions with notifications. One of the ways in which this interaction was measured was the idea that if the notification drawer is be opened, then the notification present will be regarded as more useful. The assumption made is that "a user responds to a useful notification in a meaningful manner" (Pradhan et al., 2017), however there are cases where a notification could be useful or important but it does not require any further interaction from the user once it has been raised. An example would be a calendar application presenting a notification to the user reminding them about an upcoming event. Even though the user would detect the notification and register the notification as useful, no further interaction with the notification would be required. This shows one of the ways in which attempting to discern user intent from information on how they interact with their phone is a complex process.

3.1.6.1 Feature Space

Another challenge with creating a mobile NMS is the wide variety of input feature types. Mobile devices have a wide variety of sensors and metrics which can be used to better determine the user context. Some of these measured by the study by (Pradhan et al., 2017) include "appusage, screen on/off, Wi-Fi status, headphone status, coarse location using cellular towers, battery level, notification events (post, clear, action), notification properties (time, title, id, style, modality), notification shade opening or duration (through accessibility service), ringer mode, calendar event, raw data from accelerometer, gyroscope, proximity sensors (only 10 seconds of data recorded after notification posting to save energy) and audio features of decibel and pitch …"(Pradhan et al., 2017). While it would be complex to implement a designed system which uses these features together to determine the user context effectively, a machine learning based system would be able to use all these features together due to its ability to utilize large feature spaces. For this reason, a machine learning approach for designing a mobile NMS should be considered.

3.1.6.2 Technical

A more technical difficulty is the different ways in which notifications are used by applications for functionality purposes. One example given by (Shirazi et al., 2014) is how notifications are generated by the Skype application. "When a user is voice chatting using the Skype app, for example, the app shows and updates the current duration of the ongoing call in the notification bar. The app achieves this by generating a new notification each second until the call is ended" (Shirazi et al., 2014). If these notifications were blocked or delayed by an NMS, it would make the application appear as if it was malfunctioning.

There is even disparity in how notifications are used between the same type of application. While the Skype app raises notifications throughout a call, "… the Kakao Talk app, for example, shows only a single notification when a user voice chats." (Shirazi et al., 2014). The impact of blocking or delaying notifications which are integral to the functionality of an application is an important consideration during the development of the mobile NMS.

Users can also interact with notifications in a wide variety of ways. “As suggested by Clark [10], users can respond to an interruption in four possible ways: (i) handle it immediately; (ii) acknowledge it and agree to handle it later; (iii) decline it (explicitly refusing to handle it); (iv) withdraw it (implicitly refusing to handle it)” (Mehrotra et al., 2016). The number of different interaction methods makes the act of determining user context even more complex as different notifications will require different levels of interaction. For example, an email may require opening the email application to send a response, whereas a calendar reminder may only require viewing to impart its information.
These factors mean that simply measuring one aspect of user interaction with notifications is not enough to provide a full user context for determining which notifications are useful to the user.

3.1.6.3 Privacy and Ethics

There are privacy and ethical difficulties surrounding the development of systems for user notifications. Notifications can contain sensitive information such as private message content and location data and as such require ethics permission for collection and storage.

Storage of this sensitive data requires high levels of data security and strict user access restrictions so that only individuals who are authorized to use that data have access to it. If a data breach were to occur with user notification data it would infringe on the privacy of the users who participated in data collection.

Some analyses of notifications forego using the notification content for privacy reasons. The study by (Shirazi et al., 2014) “did not collect any information about the content of notifications due to the high sensitivity of the information that can be included in notifications” (Shirazi et al., 2014). This necessity for protection of user privacy increases the difficulty in obtaining high quality notification datasets for developing NMSs.

The impacts user notification data transmission and associated ethical considerations are discussed in further detail below in section *3.5 Privacy and Ethics Considerations*.

3.1.7 Notification Delivery Time

There are contrasting results on how much impact the time of notification delivery has on the receptiveness of the user. “According to a field study with 11 co-workers by Fischer et al. [14], the user’s receptiveness is determined by message content, i.e. how interesting, entertaining, relevant, and actionable a message is. The time of delivery, in contrast, did not affect receptivity.” (Pielot et al., 2014).

By comparison, “Iqbal and Bailey [19] showed that delivering emails at so-called breakpoints, i.e. events when a person has just mentally finished a task, reduces frustration and makes users react to them faster” (Pielot et al., 2014). These two findings seem to contradict each other in how important they determine the time of delivery for notification delivery.

An explanation for this is mentioned in the results of (Mehrotra et al., 2016). “… the value of content is used for deciding whether to click or dismiss a notification. Moreover, the users very rarely state that they were busy and thus had to dismiss a notification. This could indicate that the users give precedence to a notification over the primary task, but only if the content is valuable.” (Mehrotra et al., 2016).

This further reinforces the idea that if the notification has a high enough importance to the user its delivery takes precedence over any interruption it may cause. It also highlights how low importance notifications should be managed by delivering them at times where the user is not occupied with other tasks.

3.1.8 Difficulties with Reducing the Number of Incoming Notifications

A possible approach to reduce overall interruption due to notifications is to simply reduce the number of notifications raised to the user. However, the work by (Pielot et al., 2014) emphasises that “Given that our participants typically viewed messages and social networks updates within minutes, and given that muting notifications had no effect on those viewing times, we would expect that even if these notifications are reduced, it’s likely that mobile users will check their phones more frequently to make sure that no “important” or “urgent” message has been missed” (Pielot et al., 2014). While a simple notification blocking NMS gives more user control over deciding the times when they interact with their phones, feelings of stress will most likely increase from the potential to miss important notifications as mentioned in the *3.1.5 Impact of Notifications on User Emotions* section of this literature review. For this reason, an NMS design focused on notification importance instead of notification reduction should be considered.

3.1.9 Business and User Incentives for NMSs

There is a business incentive for mobile application developers to develop or use notification management systems in their apps. In (Felt et al., 2012) it was found that in the case of a mobile app causing unwanted disruption to users such as phone vibration and L.E.D. flashing “The most common recourse was to uninstall the application” (Felt et al., 2012). The implementation of a mobile NMS will help applications to remain on users’ phones while also delivering notifications which the user finds useful.

There is also a direct user demand for development of a mobile NMS. The survey of over 400 users by (Pradhan et al., 2017) found that “92% of users check notifications on smartphones and only 25% of them are completely satisfied with the current notification systems” (Pradhan et al., 2017). This leaves a large market for a high-performance NMS.

3.1.10 Conclusions About Notifications

The ability for mobile notifications to be raised in a variety of contexts and at a wider range of times throughout the day when compared to desktop notifications gave reason for specific research into notifications form mobile devices. The large quantity of notifications received by smartphone users in combination with their potential for disruption and negative emotional impact gives a strong incentive for the development of an effective mobile NMS. In addition, there are direct user and app developer incentives for the development of an NMS due to users’ dissatisfaction with existing notification systems and a high quantity of ineffective notifications obscuring other notifications.

The development of such a system has many challenges to overcome and complexities that it needs to address. The first mentioned is the difficulty in measuring notifications and attempting to control them. There is great difficulty in determining user intent solely from the interaction users have with their smartphones and obtaining detailed notification data has significant privacy and ethics concerns.

The effect of notification importance on delivery time was also highlighted where low importance notifications should be delivered when the user is not occupied, and high importance notifications should be delivered immediately.

There was some analysis into the problems associated with a system that simply blocks notifications from a certain app or during a specific time of day where the user is busy. The possibility of missing an important notification may increase the time a user is interrupted since they are likely to check their phone more often to prevent them from missing an important notification. In addition, certain notifications are important to the functionality of apps and blocking these would result in problems in the app’s operation. For these reasons a system dedicated to managing mobile notifications has cause for development.

3.2 Mobile Notification Management Systems

3.2.1 Problems with User Management of Notifications

In determining how to develop an effective mobile NMS, currently available options should be analysed as to their effectiveness. The first of these is direct user management of notifications. While notifications can be handled by the user, these notifications "can go unnoticed when a user does not register an alert."(Mehrotra et al., 2016) and "non-persistent notifications may be forgotten about - a user riding a bicycle, might decide to attend to a notification once they arrive at the destination, yet forget to do so."(Mehrotra et al., 2016). This means that notification management should not be left solely to the user to manage themselves.

3.2.2 Currently Implemented Systems for Managing Notifications on Mobile

In (Pradhan et al., 2017), existing mobile supported user options for managing notifications are discussed. The paper mentions existing approaches to notification control such as OS-level control, do not disturb mode, and importance learning. OS level control of notifications is used by a large portion of users (42% of the 400+ users in their online survey (Pradhan et al., 2017)) and can be viewed as the default method of managing notifications. The problem with OS level control is identified by the paper as "limited flexibility" (Pradhan et al., 2017) in that there are different notification types within each individual app. The example given is the Facebook app which generates "a variety of notifications, such as birthday alerts, new message alerts, and so on" (Pradhan et al., 2017) and all of these would require separate management within the Facebook application.

Do not disturb mode disables all notifications while the setting is enabled. "22% of the surveyed users use this mode as their first choice mechanism to avoid notifications" (Pradhan et al., 2017). This system is not ideal as it removes all notifications including important ones, and as previously mentioned a direct reduction in notifications can lead to increased user stress and more frequent checking of their smartphone.

The last system discussed is one based off importance learning. These systems use machine learning to determine notification importance and raise notifications only when the importance is high enough. The issue with current implementations of these systems highlighted by the paper is that "this approach requires developers of each app to develop their own custom experience-sampling approach and a prediction algorithm for their notifications" (Pradhan et al., 2017). If each app implemented their own NMS, there would be a large variation in the effectiveness of each system due to different development environments.

An application-specific NMS would also raise privacy concerns if it attempted to access notification data outside of its app’s scope, and as a result would be less effective than a generalized system which would have access to a larger quantity and variety of notification data from multiple applications. Also if a large variety of app developer created NMSs were used, there is a higher chance that one of these would be mishandling user notification data than if a single common framework was implemented.

The problems with the above systems identify a motivation for creating a generalized mobile NMS framework for use in different applications.

3.3 State of the Art Systems

In recent years, there has been significant development in the area of mobile NMS research with a variety of different systems being proposed. This section is focused on analysing the different state of the art NMSs, describing how they operate, and discussing their effectiveness.

3.3.1 A Context and User Aware Smart Notification System (2015)

The NMS developed by (Corno et al., 2015) was created to provide features for managing notifications from a variety of sources and determining which location they should be delivered to. The focus of this system is on the optimal delivery method to reduce notifications being sent to multiple devices when the user has already seen the notification. This system is designed for distributed Internet of Things (IoT) systems, however many of the same principles for notification management and delivery still apply in the smartphone notification context.

The system operates by deciding “a) who should receive an incoming notification; b) what is the best moment to show the notification to the chosen user(s); c) on which device(s) the chosen user(s) should receive the notification; d) which is the best way to notify the incoming notification.” (Corno et al., 2015). While the system’s purpose diverges from the aim of this project since it is more focused around notifications in a general IoT space instead of a single mobile device, it does demonstrate machine learning for analysis of notifications and the use of a synthetically augmented dataset to train an NMS using different notification characteristics as the input feature space.

The study analyses three different supervised machine learning algorithms (Support Vector Machines, Gaussian Naïve Bayes, and Decision Trees) to determine which results in the best performance accuracy. The input features for these systems compose of “user personal sensors”, “environment sensors”, and “general IoT sensors” (Corno et al., 2015).

3.3.2 Interpretable Machine Learning for Mobile Notification Management (2017)

In the area of mobile NMSs, PrefMiner (Mehrotra et al., 2017) was created to automatically extract rules about notification delivery based on users’ interaction with those notifications. The rules generated are then proposed to the users for their acceptance or rejection. This is to make the system “intelligible and interpretable for users, i.e., not just a “black box” solution” (Mehrotra et al., 2017) by providing these rules in a human interpretable manner. The rules for this system are made human interpretable “by replacing each notification type with the most frequent words of the relevant notification cluster” (Mehrotra et al., 2017). As a result, PrefMiner demonstrates an implementation of interpretable machine learning on a mobile device.

PrefMiner operates by determining not only the best times to deliver certain notifications, but to also “stop notifications that are not useful, or are uninteresting or irrelevant for the user” (Mehrotra et al., 2017).

To determine which notifications to block “… the system learns the different types of interruptions that users explicitly refuse by dismissing notifications” (Mehrotra et al., 2017). From this, “PrefMiner can identify the notifications that are not useful for the users in specific situations and stop the operating system from triggering alerts related to them” (Mehrotra et al., 2017). A difficulty with using this approach alone is that by blocking notifications that are dismissed by the user, the effectiveness of certain notification types could be negatively affected. This was previously mentioned in *3.1.6 Challenges in Acquiring Notification Data and Measuring User Interactions* with Notifications where dismissal of a calendar notification after it has been raised does not signify that the notification is not important.

For this reason, the first step of the PrefMiner system identifies these “reminder notifications” which are defined as “a particular class of notifications that are always dismissed but they should be shown to users in any case“ (Mehrotra et al., 2017). These reminder notifications are identified so that rules are not created to prevent their display. One possible difficulty with this classification is that for a given new generic application, it can be difficult to identify which of its notifications are reminder notifications without knowing the details of the app’s function.

After reminder notification filtering, the second step of the PrefMiner system takes the remaining notifications and “performs clustering by considering their titles” (Mehrotra et al., 2017). This demonstrates an implementation of unsupervised learning for notification classification in an NMS.

The third and final step is to create association rules based on the input features. These features are “notification response (i.e., the user’s response to a notification), notification type, arrival time, activity, and location of the user when the notification arrived” (Mehrotra et al., 2017). Out of these features, the study “found that the association rules that are constructed by using the notification response, type and location perform better than rules constructed with other combinations” (Mehrotra et al., 2017). The conclusion is that there is “evidence that the user’s preference for receiving notifications does not depend on the activity and arrival time, but on the type of information it contains and the location of the user.” (Mehrotra et al., 2017). These “rules are constructed every day when the phone is in charging mode and not in use” (Mehrotra et al., 2017).

While it is possible that improved clustering could have been tested if notification content was used in conjunction with other features, there are corresponding privacy concerns for the system users providing this training data to the system even if the users were anonymized.

When implemented, PrefMiner takes a given notification as input, finds the corresponding rule(s) and determines the output for each rule which is whether to accept or dismiss the given notification. “The rules are extracted by calculating the ratio between the number of times X and Y co-occur and the (…) support and the (…) confidence” (Mehrotra et al., 2017). One benefit of this method is that for classes of notification with frequent occurrence, strong rules can be created with a high level of confidence in their prediction based off previous experience.

The system was designed by trying to reduce false-negatives to prevent useful notifications from being filtered. The paper mentions that obtaining a high recall value was not possible since some important non-reminder notifications were “dismissed because they do not require any further actions, such as the final message of a chat conversation” (Mehrotra et al., 2017). This is mentioned as “a fundamental trade-off in the design of this class of systems”(Mehrotra et al., 2017).

“During the 15-day study, PrefMiner suggested 179 rules to the participants out of which 102 rules (i.e., 56.98%) were accepted. Overall, around 70% of the users accepted 50% (and above) of the suggested rules. The results also show that the average number of notifications that are successfully filtered everyday is 12 (with the standard deviation equal to 8) and, thus, PrefMiner minimizes the perceived disruption for handling irrelevant notifications" (Mehrotra et al., 2017). This shows that PrefMiner is effective in reducing the number of disruptive notifications which are delivered to the user, however there is a significant level of direct user interaction required to validate the rules it generates.

3.3.3 Understanding and Managing Notifications 2017

For this study by (Pradhan et al., 2017), a supervised classification approach was taken to predict notification importance. This system operated on determining notification importance with "a combination of user-control method and a passive monitoring of user engagement with any notification to implicitly infer its importance" (Pradhan et al., 2017).

The first step that was performed was to determine the "user perceived importance" of the feedback they received from the users. Prior to this stage "explicit feedback of importance from users regarding the notifications using Snotify app [6]" (Pradhan et al., 2017) was collected.

Using this data, certain apps were seen to provide notifications which were always important and other apps raised notifications which were never important. The study proposed providing "a user with an interface where they can identify the set of apps whose notifications are always (or never) important for them." (Pradhan et al., 2017). This simplifies the notification classification problem significantly as notifications from these apps can be classified simply into immediately reject (never important) and accept (always important). One downside of this method is that direct user interaction with the system is required, and changes in user context are not automatically accounted for. An example of where this could be problematic would be if a user decided that notifications from a region-specific app should always be blocked since the user is planning to travel abroad, but upon returning they forget to unblock the given app's notifications.

The remaining apps with notifications of varying importance are analysed using a rule-based system. The notifications with neutral responses were removed from the dataset, as well as the notifications already covered by the per-app classification mentioned above. From the remaining notifications, a "rule-based classifier that classifies a notification as important if one of the following conditions is met" (Pradhan et al., 2017) was created. The rule-based classifier had an accuracy of 86.29%.

The second step performed was to extract the features to use for training the machine learning system. A set containing 22 different features which are grouped into different types. The feature types are temporal, user device activity, location, sensor data, and notification-based features. Each of the 22 features were then ranked "based on the information gained by adding it for predicting the notification importance" (Pradhan et al., 2017).

The feature ranking "results show that the temporal features like hour of the day or temporally local event-based feature like last app use are the most important features. Furthermore, notification property driven features, location and calendar events are also relatively important features in terms of information gain results. However, activity level or sensor based features are not that important …" (Pradhan et al., 2017). This is in contrast with the PrefMiner system which found that notification response, type and location were the most important features and that temporal data was not as effective in determining user notification preference.

The third step performed was to train the machine learning system. Multiple machine learning algorithms were used to determine which algorithm was most effective. The different types of algorithm were "Random Forest with 100 trees, Decision Tree, Support Vector Machine (SVM), and Linear Regression models for predicting notification engagement classes, i.e.interacted and ignored." (Pradhan et al., 2017).

Evaluation of the systems was conducted by its ability "to predict two classes *interacted* and *ignored*" (Pradhan et al., 2017). This ability was measured through measuring the accuracy, precision, recall and f-score of the overall predictions. A "k-fold cross validation approach with k = 10" was used for each user.

The results found that the "Overall average accuracy of prediction is more than 87%. The result also shows that precision and recall are more than 87%" (Pradhan et al., 2017). "Random Forest performs best in terms of all four metrics and also shows comparably less variation across users. On the other hand, decision tree model performs best in terms of time without compromising too much on accuracy. Therefore, it has been selected for our smartphone prototype implementation. In comparison, SVM is the slowest." (Pradhan et al., 2017). This study emphasises the priority of faster processing over slight improvements to the evaluation metrics. This priority is important for developing machine learning systems for deployment on mobile devices.

The paper then compares the performance of systems trained on three different formats of user data: user's personal data, multiple users' data and clustered user data. The user data is clustered using the k-means algorithm with features of "number of applications used, number of unique locations visited, number of notifications received, and number of each engagement levels" (Pradhan et al., 2017).

3.3.4 C-3PO: Click-sequence-aware DeeP Neural Network (DNN)-based Pop-uPs RecOmmendation

While the previously mentioned notification management systems have been focused on improving user experience by reducing the impact of unwanted or unimportant notifications, a very recent system by (Huang and Kao, 2019) has been developed with an advertising focus.

The aim of this system was to increase the click-rate of advertising notifications and the retention rate for the corresponding apps. The retention rate is defined as "the number of users that logs in the app at least once in the following seven days over the number of new users on that day" (Huang and Kao, 2019).

The system also raises notifications to inform users of "their smartphone operating condition", such as high phone temperature and junk files, so that the system "can increase the open rate" of the notification drawer, resulting in the display of more advertising notifications.

The system operates by taking input features from the user's phone about their current context and phone state, sending the feature data to a remote server to train a Deep Neural Network (DNN), and uses this trained model to determine a score for a given notification which determines whether it is displayed or not.

One downside of this architecture is that due to the large feature space, their system specifications and the implementation of a DNN, there is a high hardware requirement to train this network. This means that the system cannot be implemented on-device, and as such there is a "need to upload the value of the features from client application" (Huang and Kao, 2019), instead of storing the notification features locally. This results in a privacy concern since it requires transmission of highly sensitive notification data, and storage of this data on a remote server.

While the implementation decreases notifications that were deemed troublesome to users, this was implemented primarily to help "increase the click-through rate of push notifications/pop-ups” (Huang and Kao, 2019) for advertising by removing unwanted advertising notifications instead of being implemented to improve user experience.

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 click-rate of notifications. It demonstrates how a mobile NMS can be used to implement user personalized advertising through mobile notifications, in a similar manner to existing forms of online personalized advertising.

From an ethics perspective, it will be very important in the future to distinguish NMSs that are intended for user experience and NMSs which are intended for increased advertising effectiveness.

3.4 Comparison of SoA Systems

The table below summarizes the main features of the state-of-the-art systems discussed. It displays whether real user data was used for the system, if the system operates in real time, if the system was deployed on a mobile device, and the type of machine learning used.

Table 1: State-of-the-art mobile NMSs and their features

	Real User Data?	**Real Time Analysis?**	**Deployed on-device?**	**Type of ML used?**
A Context and User Aware Smart Notification System (Corno et al., 2015)	Synthetic information added to augment real user dataset	Separate training and classification stages. Training can take significant time depending on algorithm used.	No. Using Python script	Supervised (Support Vector Machine (SVM), Gaussian Naïve Bayes, and Decision Trees)
PrefMiner (Mehrotra et al., 2017)	Yes	Rules constructed when not in use. Rules implemented in real time.	Yes	Rule-based (with unsupervised learning for notification clustering)
Understanding and Managing Notifications (Pradhan et al., 2017)	Yes	No	No. Uses Weka, Matlab and vowpal-wabbit	Supervised (Random Forest, Decision Tree, SVM, Linear Regression)
C-3PO (Huang and Kao, 2019)	Yes	Yes	No. On remote server	Supervised (Deep Neural Network)

As can be seen by this table there are no NMSs that have been implemented using reinforcement machine learning with most opting to utilise supervised machine learning. In addition, there are no systems implemented using exclusively synthetic data, with the closest being the system by (Corno et al., 2015) which uses feature synthesis to augment existing data. This shows a gap in research in both the areas of NMS training on synthetic data and

NMSs implementing reinforcement learning. In addition, out of all of these systems, Prefminer was the only system which was deployed fully on-device for mobile.

3.5 Privacy and Ethics Considerations

One of the future goals for NMSs is to have a system that can be deployed on mobile devices directly. The reasoning for this is that an on-device mobile NMS would require no transmission of sensitive mobile notification data to a remote server and would not require storage of that data on that server.

Another difficulty with transmitting sensitive data from a mobile device is the difficulty in determining whether sensitive data is being transmitted to malicious applications. Applications such as AppIntent (Yang et al., 2013) highlight the difficulty in identifying whether transmitted data is user intended or is a privacy breach. By implementing an NMS which does not transmit user sensitive data, this problem is avoided.

3.6 Synthetic Notification Datasets

A synthetic notification dataset was used for the training and evaluation of the machine learning system developed as part of this project. The creation and evaluation of this dataset is described fully by (Fraser, 2018).

3.6.1 Utility and Benefits of a Synthetic Notification Dataset

Due to the privacy concerns surrounding the collection of "in-the-wild" user notification data, there are "there are few, if any open-source mobile notification datasets in existence"(Fraser et al., 2017) and those that do exist are usually restricted to exclusive research use and specific institutions.

One of the benefits of using a synthetic notification dataset is that it reduces some of the impact on users' privacy since the direct content of the notifications is not used. By generating data based on the main features of the notifications such as app, category of app and time of day, many of the privacy sensitive notification details such as message content and user location are not divulged in the synthetic dataset. This provides a dataset with a reduced impact on user privacy.

However, by implementing a synthetic notification dataset, there is a trade-off between the amount of privacy and the detail of the notifications. Since notification content, location data, names of users and other similarly privacy sensitive notification features are not available in the synthetic notification dataset, the set of features available to the machine learning system is reduced.

A synthetic dataset also allows a larger quantity and variety of data to be generated, however the variety of data is dependent on the generation method and the diversity of the dataset used to create the synthetic data.

3.6.2 How "In-the-Wild" Data was Collected

The dataset by (Fraser, 2018) was created from "in-the-wild" notifications collected from mobile users. This collection of notification data was conducted using the WeAreUs app (Fraser, 2019). The app used two methods for collecting user data, a "background-sensing method" and an "experience sampling method". The background-sensing method used the Android SDK's Notification Listener Service (Google LLC, 2019a) and hourly logging of the mobile device's sensor states to collect relevant data without direct interaction with the user. Most of the data obtained was from this method of collection.

The experience sampling method used more active interaction with the user to obtain information. This included prompting users to answer questions "pertaining to the notification and their current context" (Fraser, 2018) after interacting with a notification and prompting users to "answer questions regarding their current state-of-mind and their immediate context" (Fraser, 2018) when they unlocked their screen. While this data makes up the minority of the data collected, this data contains higher quality details regarding the user's context and their reaction to the notifications.

These data sampling methods resulted in an "in-the-wild" dataset from "15 participants (2 Female and 13 Male), ranging in ages from 21 to 64" (Fraser, 2018) containing "Over 30,000 in-the-wild notifications […] as well as 4,940 smartphone general-usage logs and a total of 291 ESM questionnaires […] answered by participants" (Fraser, 2018).

3.6.3 Data Generation

A Generative Adversarial Network (GAN) was used to generate the synthetic notifications by using the WeAreUs dataset as the real data. Prior to implementing the WeAreUs dataset, the users of the WeAreUs app were surveyed about their notification behaviour. "The responses show that in 27 of the 41 cases (65.6%), participants believed their behaviour of engagement toward the notification was good" (Fraser, 2018). The other "34.4% embodied poor behaviour" (Fraser, 2018) and so were filtered out of the dataset before it was used for the GAN.

Synthetic notifications were created by the generator component of the GAN, and the discriminator component was "given alternating real and generated values"(Fraser, 2018). The aim is for the synthetic and real data to be similar so that the discriminator cannot differentiate between which data is real or synthesised. At this point, the generator "can be used to generate data which mimics the distribution of real data"(Fraser, 2018) and so can create a realistic synthetic mobile notification dataset.

3.6.4 Evaluation of this Dataset

The dataset was evaluated by training a neural network and a random forest classifier on the synthetic data and evaluating their performance when classifying real world data using the metrics of accuracy, precision, recall and F1-Score. A random classifier was also used for comparison. "While neither classifier was able to reach the same performance height of those trained using real data, there is still value in the predicted results with performance nearing 70%"(Fraser, 2018) across all performance metrics. The random classifier's performance was around 50%.

3.6.5 Possible Downsides of Using a Synthetic Dataset

The downsides mentioned by (Fraser, 2018) describe how subtle nuances in the data were lost. "… it is clear that the synthetic dataset is less nuanced than the real dataset. The number of unique subjects, places and apps found in the dataset are much lower than that of the real dataset suggesting that the generative model was unable to learn a holistic view of these features. However, the top subject/apps/places were still identifiable and their engagement rates generally accurate, hence the dataset is an adequate representation of real world data for the purposes of this study" (Fraser, 2018). This reflects the trade-off previously mentioned between the level of precision possible and the amount of privacy specific information divulged.

3.6.6 Conclusion

The slight performance trade-offs displayed in the evaluation of this dataset were considered to be worth the large improvement in privacy consideration and data accessibility compared to a similar "in-the-wild" dataset, hence this synthetic dataset was used for this project.

4 Methodology

4.1 Machine Learning Systems

As mentioned in the literature review, there has been little research conducted in the area of reinforcement machine learning for mobile NMSs. In addition, purely synthetic datasets have not been used for training and evaluating mobile NMSs in the literature discussed. The following sections describe the methodology used for implementing the reinforcement learning algorithm and the evaluation of the system.

4.2 Reinforcement Learning Algorithms

4.2.1 General Design

The system design for a reinforcement learning system follows the general layout shown below (Figure 1). It operates based on the interactions between the agent and the environment. The agent performs an action on the environment and the environment returns what the environment has changed to after that action (State, S_t) and how effective that action was in working towards the system's overall goal (Reward, R_t). The overall aim of the reinforcement learning system is to maximise the overall reward by choosing the ideal actions based on the previous environment state. In other words to create a policy that the agent should follow in order to choose the optimal actions for a given state. This implementation of states, actions and rewards is referred to as the Markov Decision Process.

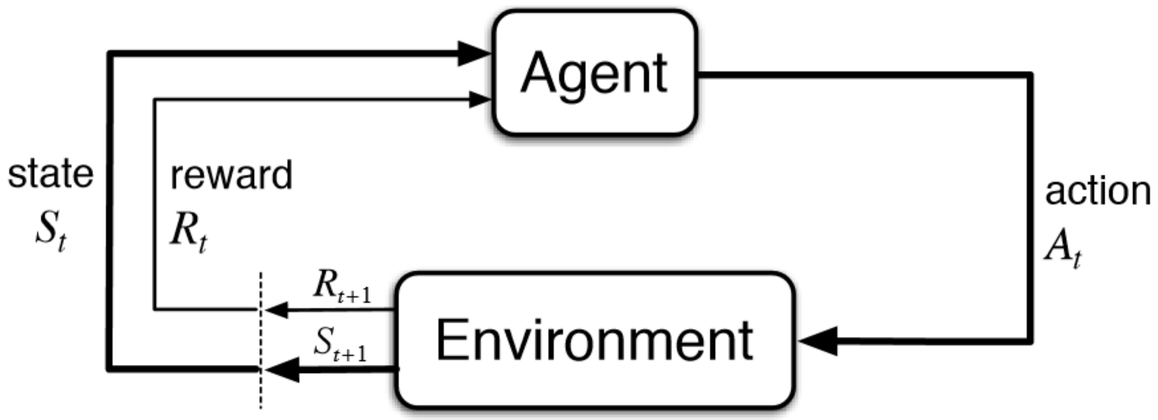

Figure 1: Agent-Environment diagram of a reinforcement learning system (Ashraf, 2018)

Different reinforcement learning algorithms were considered as to their effectiveness for the project. First, it was determined whether a model-based or model-free algorithm should be used. Then

4.2.2 Model-Based vs Model-Free

The two types of reinforcement learning algorithm are model-based and model-free. Model-based algorithms require some way of determining which states are going to occur in the future and how they will react to chosen actions. Model-free algorithms are based on environments where their reaction to different actions and the different state transitions that occur cannot be predicted.

Since this project involves the prediction of user actions based on mobile notifications, it is impossible or infeasible to create a model which determines the next notifications which will be raised based on a current action and state. As a result, a model-free system must be used for this project.

Popular model-free systems include Monte Carlo reinforcement learning, Temporal Difference learning and Q-Learning.

4.3 Types of Reinforcement Learning Algorithms

4.3.1 Monte Carlo

Monte Carlo Reinforcement Learning operates on the idea that by taking enough random starting states, taking actions for each of these states until some defined end and seeing the corresponding rewards for the actions for a given state, a policy can be determined. In the game example given in a blog by (Salloum, 2018) the Monte Carlo system plays "an episode of the game starting by some random state (not necessarily the beginning) till the end, record the states, actions and rewards that we encountered then compute the V(s) […] for each state we passed through" (Salloum, 2018) where V(s) is the value function

While other algorithms involve the calculation of multiple state-action paths, Monte Carlo requires the execution of these paths until some end state is reached and does not update the system until those end states are reached.

This is not optimal for implementing a mobile notification management system since notifications will be received on a constant basis and a system which could update in real-time without retraining on the full dataset would be ideal. Users interact with their phones on a continuous basis and there is no end state for this interaction.

Another issue is that this method can miss possible states and actions in determining its policy due to its random nature in choosing its initial states. This means that the final implemented policy for action estimation may be missing information on which action to take for certain states.

4.3.2 Temporal Difference and Q-Learning

Both Q-Learning and Temporal Difference learning are similar in their implementation in that they calculate a Q-value which determines for a given state which action should be taken. However, for a given state Q-Learning updates the Q-value for only the previous state whereas Temporal Difference learning updates the Q-values for all states in past steps. By only updating the most recent state, there is a weaker dependence on the order in which notifications arrive in training.

A Q-Learning approach was chosen for this project over Monte Carlo due to its ability to guarantee the incorporation of all of the training data in the dataset. Q-Learning was chosen over Temporal Difference learning since the reduction in order dependence simplifies the system's analysis by not having training notification order as a system parameter with strong influence on the system. This simplifies the implementation of evaluation mechanisms for the system.

There is also a version of Q-Learning called Deep Q-Learning which can be implemented to compare the two system types as to their performance.

In this project, the environment was implemented as an OpenAI Gym (OpenAI, 2019) environment through a Python script and the Q-Learning/Deep Q-Learning agent was implemented as a Python script which interacts with that environment. The code for both agents can be found at (Sutton, 2019b) with the corresponding Gym environment at (Sutton, 2019a).

4.4 Q-Learning Implementation

4.4.1 State, Action and Reward

The Q-Learning algorithm follows the Markov Decision Process as shown in (Figure 1). The states for the Q-Learning and Deep Q-Learning implementation used in this project were the different possible combinations of feature values from the synthetic notifications. For example, if the features of app package and app category were used with 3 possible app packages in the dataset and 4 possible app categories, there would be a total of 12 notification states.

The action for each state was defined as whether the user interacted or did not interact with the notification. Each notification in the synthetic dataset contained a feature which stated whether that notification had been interacted with by the user or not. The action space of user interaction was used since it had this corresponding ground truth value in the synthetic dataset which could be used for determination of reward values and evaluation of performance. While it was previously mentioned how the direct blocking or acceptance of notifications based on a single aspect of user interaction has a variety of associated problems, this action space could be expanded on in the future to incorporate more complex actions such as interaction with high importance or messaging notifications.

The reward values for the system are determined by comparing the action value predicted by the agent for a given notification to the action value stored in that notification. If the two match, then a positive reward signal of "1" is sent from the environment to the agent. Otherwise a zero reward signal is sent.

4.4.2 Exploration vs Exploitation

Initially the reinforcement learning algorithm will have no information on which action to take, however over time it will learn which actions give a higher reward for a certain state. In training both Q-Learning and Deep Q-Learning systems it would be ideal if the trained system was incorporated into the training over time. For this reason, there is an "exploration vs exploitation" trade-off where the initial stages of training will have the system choose random actions (exploration) and over time these random actions will instead be replaced by the actions predicted by the system (exploitation).

The value epsilon is used to denote the proportion of exploration actions to take versus the number of exploitation actions to take. A larger epsilon means that for a given state, there is a higher probability that a random exploration action will be taken instead of a trained system exploitation action. The training of these systems is organised so that the initial value of epsilon is 1 and this value decays exponentially as more training steps are performed. These training values are shown in the later *System Training Metrics* sections in section *5. Findings*.

4.4.3 Q-Tables

Q-Learning operates by creating a Q-Table (Figure 2) where each column represents a different action in the action space, and each row represents a different state in the state space. The values stored in each cell of the table represent how strong the system believes that the action at that column will maximise overall reward given the state for that row. From a trained table of these Q-values, the Q-Learning algorithm can determine what the optimal action to take is for a given state. An example of this is shown in Figure 2 below where if the state corresponding to the second row of the Q-Table (S_1) was presented to the trained Q-Learning algorithm, then the system would choose the action corresponding to the second column (A_1) since it has a larger Q-value than the action at column one (A_0).

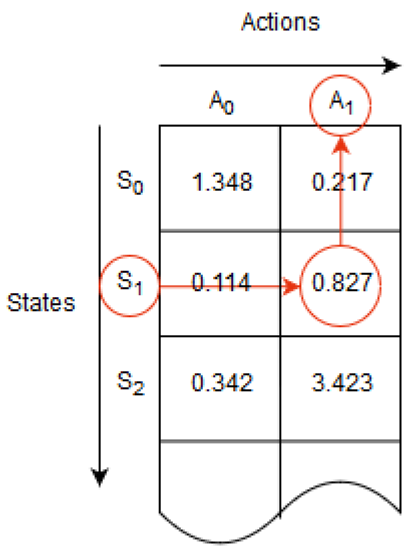

Figure 2: Visualization of the Q-Table decision process given state S_1

For this project, the Q-table had two columns, one to show no user interaction with the notification and the other for user interaction. The number of rows was dependent on the combination of notification features.

These Q-values are determined by using the Bellman Equation (Figure 3) which determines the new Q-value ($NewQ(s, a)$) in the Q-Table for the current state (s) and the action to take (a). The action (a) is determined randomly if exploration is currently being used (i.e. a random number between 0 and 1 is less than epsilon) or from the Q-Table process shown above if exploitation is being used (i.e. the random number is greater than epsilon).

The formula also takes the previous Q-value at "s, a", the reward ($R(s, a)$) from the environment for taking the action (a) for the state (s), and the maximum expected Q-value for the next state (s') out of any possible actions for that state (a'). This maximum expected Q-value for s' can be determined by finding the maximum Q-value in the s' row of the Q-Table.

$$\underbrace{NewQ(s,a)}_{\text{New Q value for that state and that action}} = \underbrace{Q(s,a)}_{\text{Current Q value}} + \underbrace{\alpha}_{\text{Learning Rate}}[\underbrace{R(s,a)}_{\text{Reward for taking that action at that state}} + \underbrace{\gamma}_{\text{Discount rate}} \underbrace{max\,Q'(s',a')}_{\text{Maximum expected future reward given the new s' and all possible actions at that new state}} - Q(s,a)]$$

Figure 3: The Bellman Equation (Simonini, 2018b)

To implement this algorithm, system training was performed for 1000 training episodes. Each training episode involved the training methodology described above and used every notification in the chosen synthetic notification dataset once. Between episodes, the order of the synthetic notifications presented by the environment to the agent was randomized. A learning rate of 0.7 and a discount rate of 0.618 was chosen for this project's implementation.

In a mobile implementation of this system, more notifications could simple be appended to the dataset before a training episode to implement those notifications in real-time.

4.4.4 Testing Implementation

For testing the trained system, the process described above for Figure 2 was used after being presented with the training notification dataset. Due to the different possible orders of state paths, 100 episodes were used for testing the algorithm however in practice there was little difference in performance between episodes.

4.5 Deep Q-Network Implementation

4.5.1 Difficulties with Q-Tables

As can be seen for the Q-Learning example, the size of the Q-Table can become very large for a high number of notification features and number of feature values. For implementations with n features and approximately m different values for each feature, the complexity of this algorithm and number of rows in the Q-Table is approximately O(n×m).

A possible solution is to use a Deep Q-Network instead. This in effect replaces the Q-Table with a Deep Neural Network which takes in a state and outputs the possible actions with their corresponding Q-values. A trained system would then perform the action with the largest Q-value.

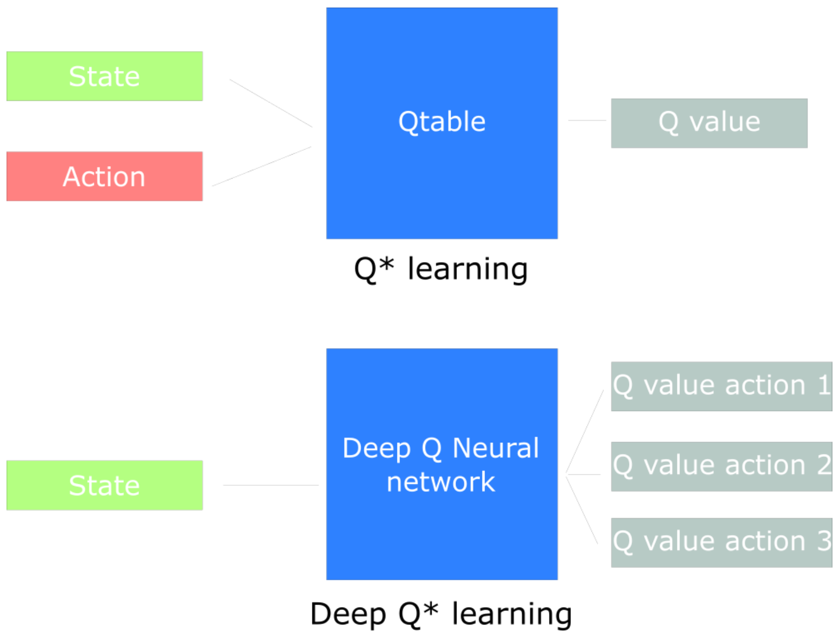

Figure 4. Differences in the overall system structure between Q-Learning and Deep Q-Learning (Simonini, 2018a)

Each possible notification state is one-hot encoded so that input notifications can be encoded and used for the neural network.

4.5.2 Huber Loss Equation

In order to train this system, a loss equation is required for training the neural network weights. The loss equation used for this project was the Huber Loss function (Figure 5) which takes the difference between the predicted (y) and actual ($f(x)$) values for the reward for a given state-action pair and outputs the corresponding loss value to be minimized by the neural network.

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

Figure 5: The Huber Loss function

The delta value determines the cut-off for the piecewise function. This project used a delta value of 1. While training episodes were used for the Deep Q-Network (DQN) training, only ten of these episodes were used instead of the 1000 used for the Q-Table. This was done to achieve comparable performance times between the two algorithms.

4.5.3 Replay cache

Since the neural network will decrease the loss value based on the most recent notifications used for training, past changes to the network's weights can be “overwritten” by training on more recent notifications. This is not a desirable effect as it means the network “forgets” the training on the initial sets of data and instead fits more strongly to recent notifications.

To combat this, a replay cache is used where past notifications are stored in a cache and every time a new notification is used to train the network, some notifications are randomly selected from the cache to train the network as well. The replay cache batch size determines how many notifications are chosen from the cache per new notification.

4.6 Evaluation Methodology

4.6.1 K-Fold Cross-Validation

The systems implemented a k-fold cross validation approach for the input datasets. The purpose of this was to reduce the amount of bias in measuring system performance metrics such as accuracy and recall.

K-fold cross validation operates by splitting the dataset into k parts and selecting the first part to be the testing data. The rest of the data is used to train the machine learning algorithm and the trained system is then evaluated on the selected testing data part. The algorithm then iterates to the next part where the second part of the initial dataset now becomes the testing data and the remainder becomes the training data. This continues until all parts of the dataset are used as testing data. Figure 6 below shows this process for a k value of five.

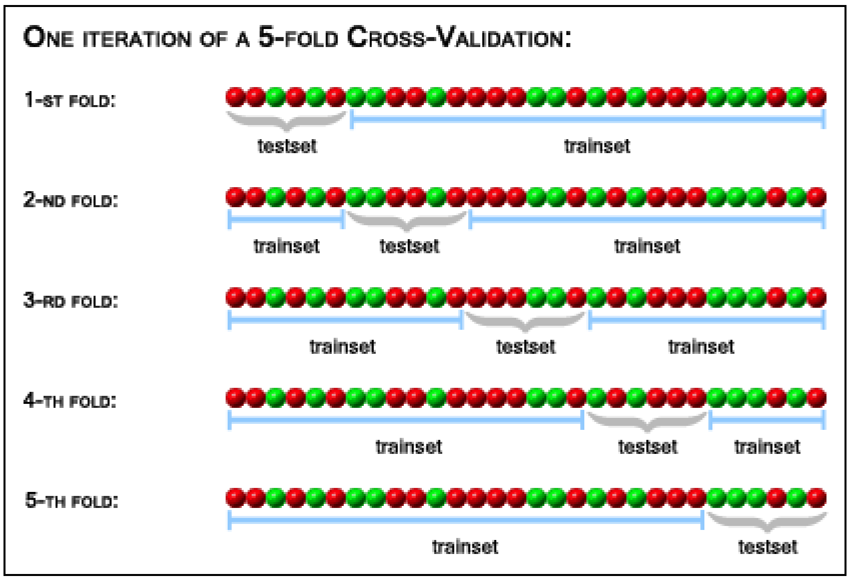

Figure 6: Diagram of 5-Fold Cross-Validation over an entire dataset (Graz University of Technology, 2006)

Provided that the dataset used is representative of the data encountered in the system’s implementation, k-fold cross-validation reduces bias in the data. For example, if there is a small streak of one type of data in the dataset, training solely on that data will result in poor performance when it is tested against the rest of the data.

The choice of k-value is important as "… there is a bias-variance trade-off associated with the choice of k in k-fold cross-validation." (James et al., 2013).

"Typically, given these considerations, one performs k-fold cross-validation using k = 5 or k = 10, as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance" (James et al., 2013).

The system developed by (Pradhan et al., 2017) also uses this method to "… evaluate the data-driven prediction models by testing with the k-fold cross validation approach with k = 10" (Pradhan et al., 2017). For these reasons, a k-value of 10 was used for this project's cross-validation.

4.6.2 Metrics Used

For evaluating the performance of the system in terms of its ability to correctly predict the users' interaction with the notifications the metrics of precision, accuracy, recall and F1 score were used.

$$precision = TP/(TP + FP)$$

$$accuracy = (TP + TN)/(TP + FP + TN + FN)$$

$$recall = TP/(TP + FN)$$

$$F1\ Score = 2 * precision * recall/(precision + recall)$$

These metrics were measured for each k-step in the 10-fold cross validation evaluation and were averaged across the 10 k-steps for brevity and presentation purposes. In order to encapsulate any differences or variation in measurements between different k-steps, the standard deviation was also calculated for these metrics across the 10 k-steps.

For evaluating the computational performance of the system, the time taken for the system to fully train and to evaluate the system were both measured using the timeit library in Python.

Information regarding the systems 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 system's 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 system which are chosen randomly instead of using the machine learning system.

4.7 Data Class Imbalance

The initial problem encountered from running the system was that there was significant class imbalance in the data, with most of the data having no user interaction. To deal with this, either oversampling or undersampling was needed.

Oversampling operates by generating extra data of the underrepresented class in order to have a class balanced dataset. However, if oversampling were to be used for this project then there would be the issue that synthetic data is being generated from synthetic data using a separate process. This would further distance the data from being similar to the “in-the-wild” equivalent dataset.

The alternative is undersampling which reduces the number of samples of the overrepresented class. Since a synthetic dataset is being used, more synthetic data can be generated and then undersampled to obtain a class-balanced dataset of a desired size. This is a benefit of synthetic datasets in that they can create as much data as desired, however care needs to be taken in generating large datasets as the diversity of notifications is based on the diversity of the “in-the-wild” dataset used in the generative adversarial network.

The end result of this is a synthetic dataset with approximately the same number of notifications with direct user interaction as there are notifications with no user interaction.

Undersampling would not be used in practice to balance the datasets since there is a low amount of user data available and the system would want to maximise its utilization of the user dataset. More likely that a synthetic data generation process similar to the one used for this dataset would be used to oversample. While it would be simpler to replicate or retrain on notifications of the less common class, this would introduce strong bias towards that action value for notifications with those notification feature values.

This shows a disparity in the way in which synthetic and real-world data can be treated for this type of system.

5 Findings

5.1 Size of Dataset Used

One factor in evaluating the performance and behaviour of the two algorithms is the size of the notification dataset which they use for training and testing. The analysis of performance versus dataset size is an important factor as it corresponds to the number of notifications required by the system for it to start achieving a high level of classification performance.

Note that the dataset sizes shown are the sizes of the overall notification dataset used in the 10-fold cross-validation. As a result, the training and testing set sizes are 90% and 10% of the overall dataset size respectively. In addition, all tests for these systems were performed on an Intel i7-7700HQ 2.8GHz CPU with 16GB of system RAM.

5.1.1 Machine Learning Evaluation Metrics

5.1.1.1 Q-Table

The plot below (Figure 7) shows how precision, accuracy, recall and F1 Score are affected by the number of notifications in the notification dataset used for 10-fold cross-validation of the Q-Table system. The standard deviation of these results is shown below in Figure 8.

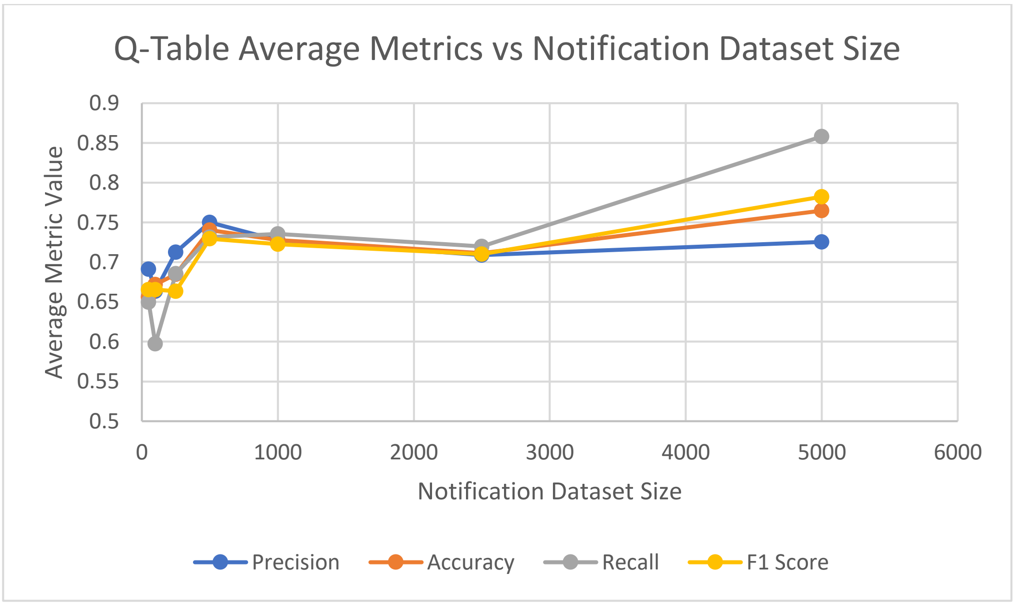

Figure 7: Average metrics across 10-fold cross-validation for the Q-Table implementation and their change with dataset size

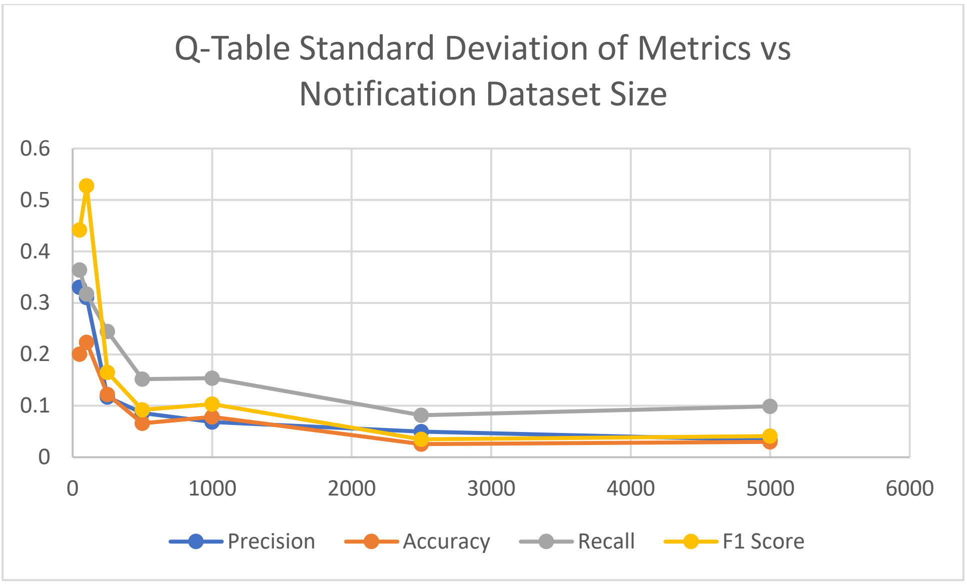

Figure 8: Standard deviation of the metric results from Figure 7

5.1.1.2 Deep Q-Learning

As with the Q-Learning algorithm, the Deep Q-Learning algorithm was evaluated on the same metrics of performance for various dataset sizes.

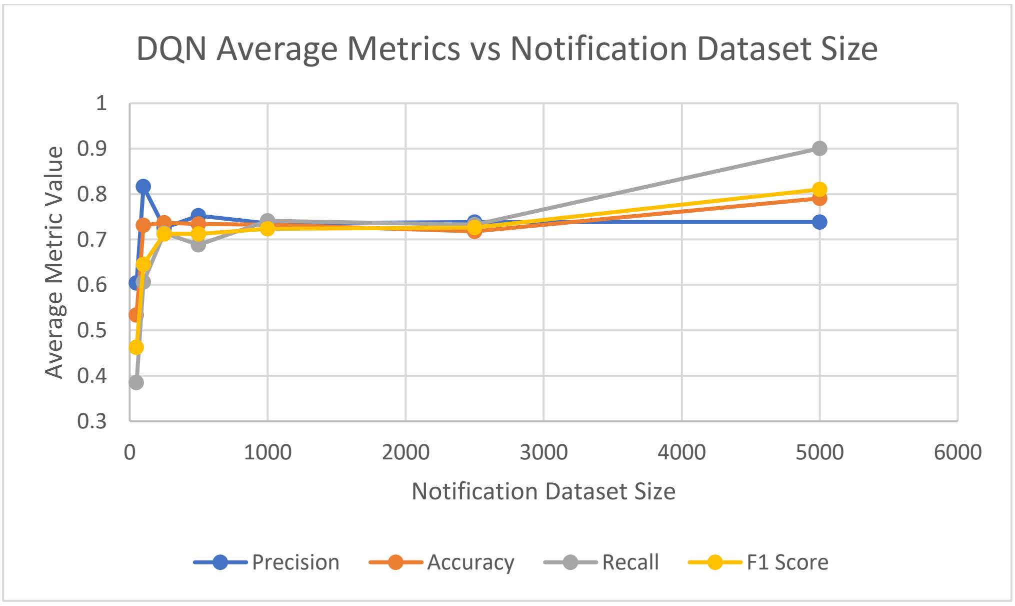

Figure 9: Average metrics across 10-fold cross-validation for the DQN implementation and their change with dataset size

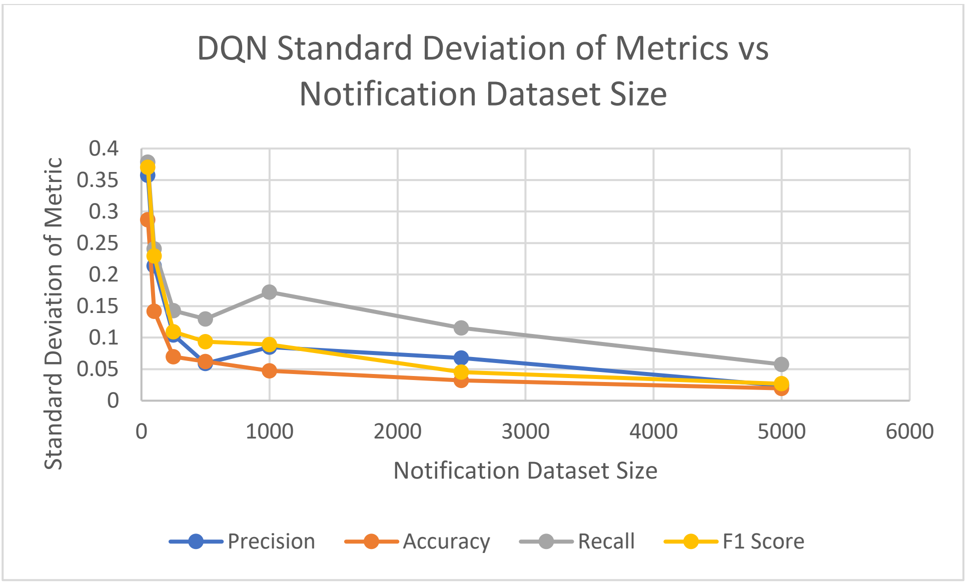

Figure 10: Standard deviation of the metric results from Figure 9

For dataset sizes below 500 notifications, there were lower values for all metrics in both systems. In particular, the DQN system found very low recall and F1 Score values which indicates that for low numbers of training notifications, the DQN system was predisposed towards guessing the negative action (no user interaction) for most notifications when implemented.

The performance of both systems seems to level off from 500 to 2500 notifications after which it increases in the metrics of recall, F1 Score and accuracy for 5000 notifications.

For 5000 notifications, the Q-Table system displays a high accuracy of 76.5%, a very high recall value of 85.8% and high F1 score of 78.2%. The DQN system showed even better results with an accuracy of 79.1%, a recall of 90.1% and an F1 Score of 81.0%.

There were high standard deviation readings for all notification datasets below 250 notifications which indicates that the training sets used for these dataset sizes were not representative of the overall testing dataset, leading to high variation in performance between different k-steps.

5.1.2 Computational Performance Metrics

The training times (Figure 11 and Figure 13) and testing times (Figure 12 and Figure 14) were measured for these algorithms to give an indication of how the computational performance changed with the size of the notification dataset.

5.1.2.1 Q-Learning

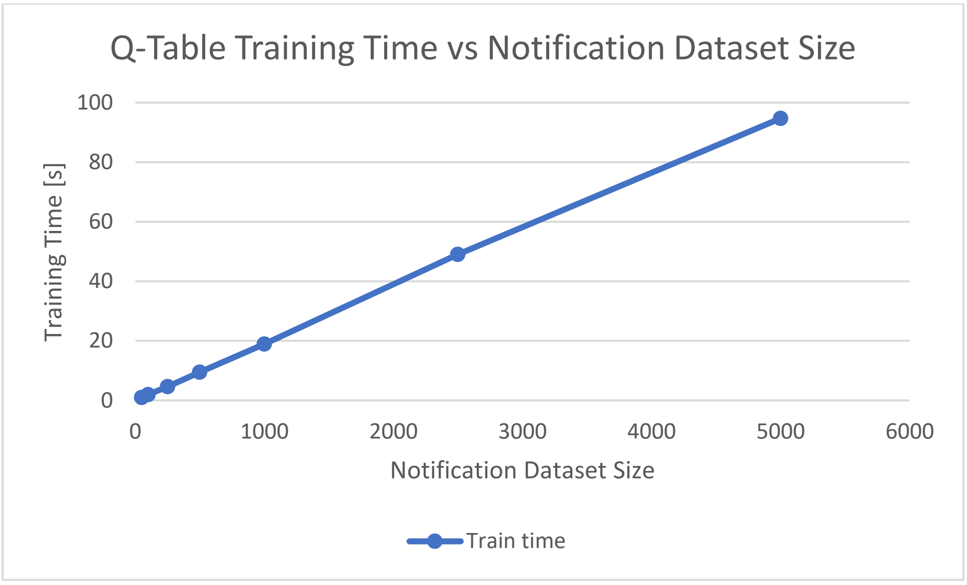

Figure 11: The effect of notification dataset size on system training time

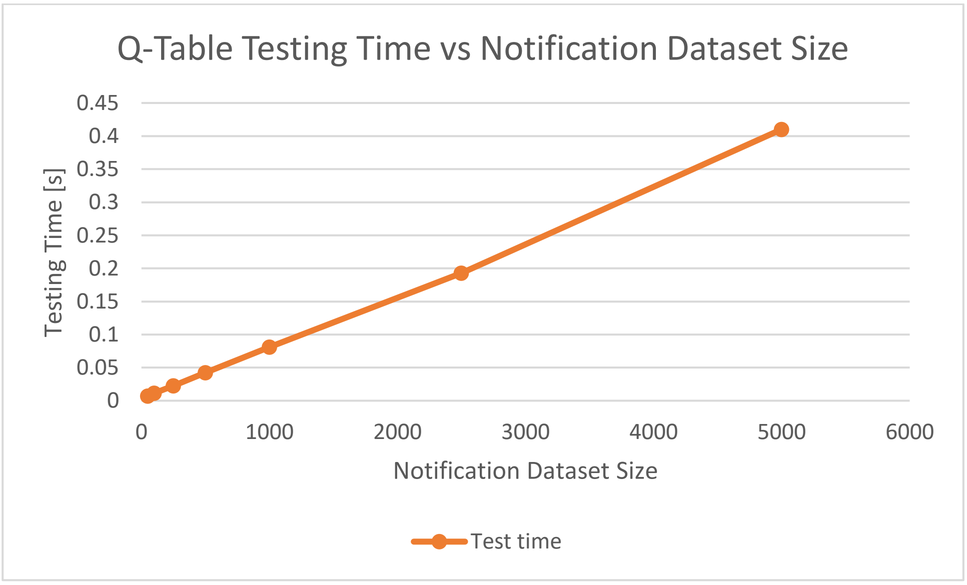

Figure 12: The effect of notification dataset size on system testing time

5.1.2.2 Deep Q-Learning

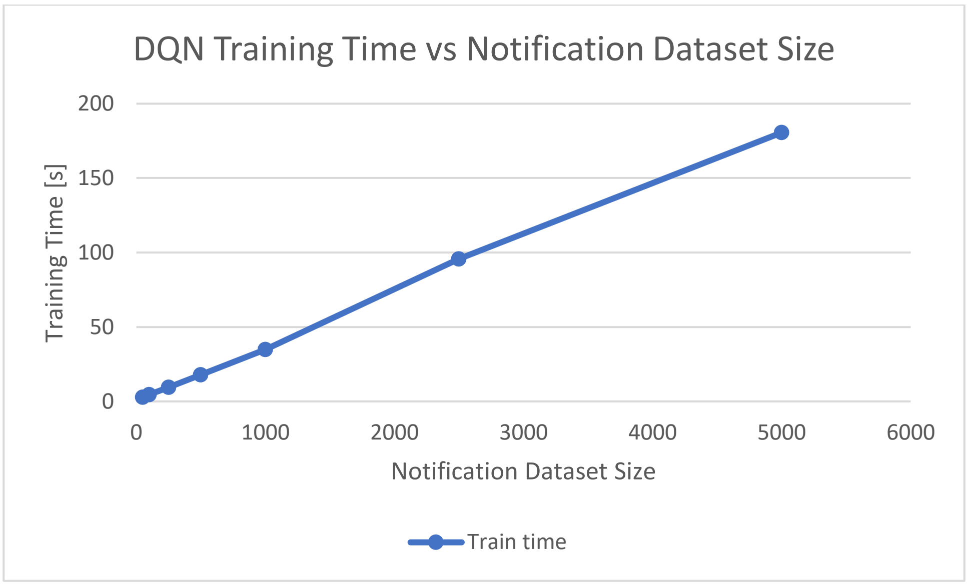

Figure 13: The effect of notification dataset size on system training time

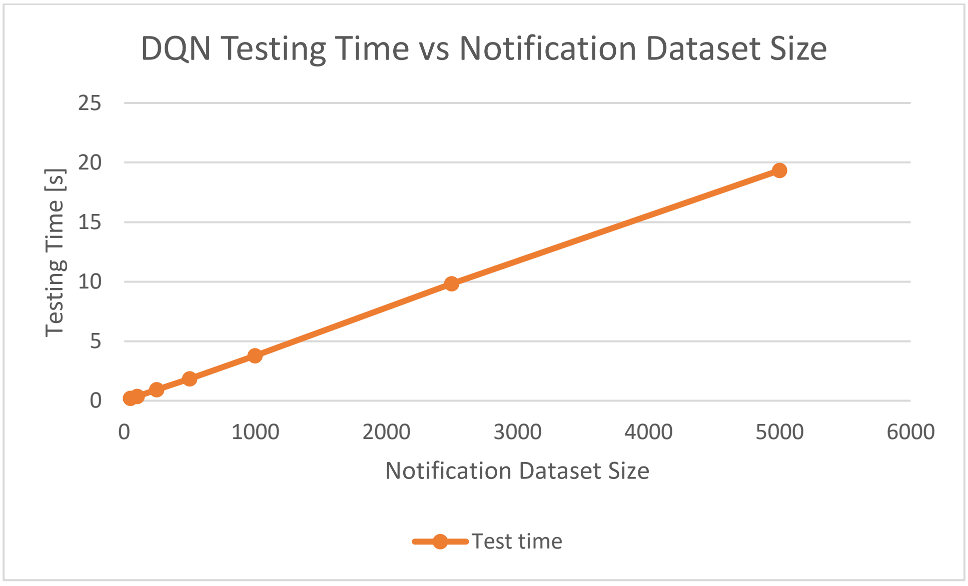

Figure 14: The effect of notification dataset size on system testing time

For both systems there is a linear relationship between the size of the notification dataset and the time taken to train and implement the system. While the training time for the DQN system was around twice as long as the Q-Table system, there was a higher ratio difference in the two systems' testing times with the Q-Table system running around 40 times faster than the DQN.

5.1.3 System Training Metrics

5.1.3.1 Q-Learning

Below are graphs showing the epsilon and percentage reward values during the training process. These metrics were taken for each episode of the 1000 training episodes used. They were measured to give insight into the training process of the system and how this changes with dataset size.

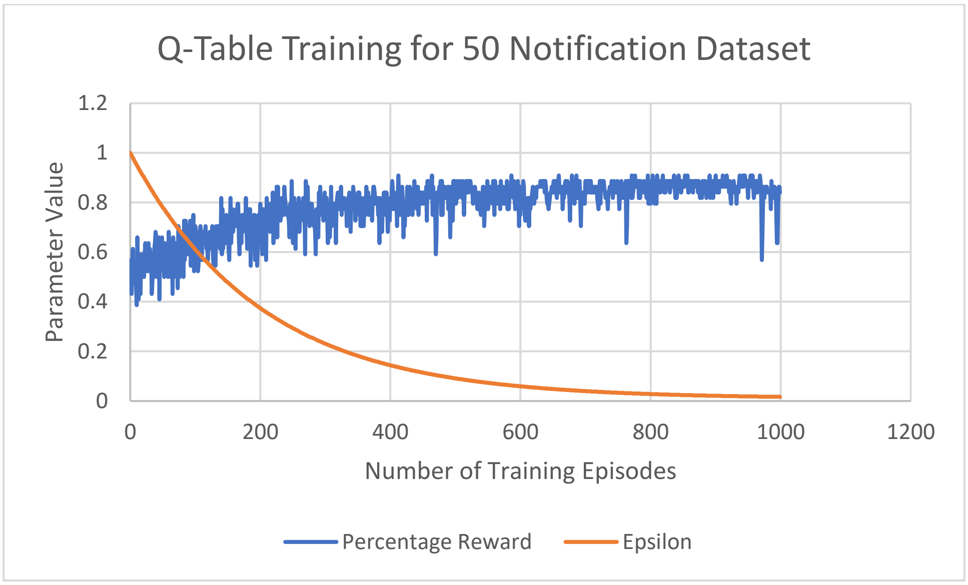

Figure 15: Q-Table training parameters for 50 notification dataset (5 training notifications)

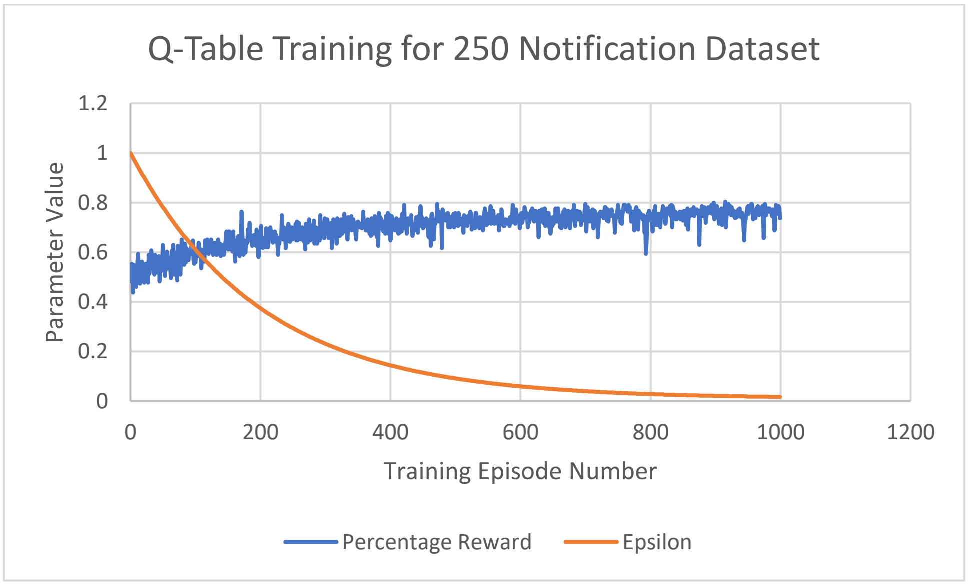

Figure 16: Q-Table training parameters for 250 notification dataset (25 training notifications)

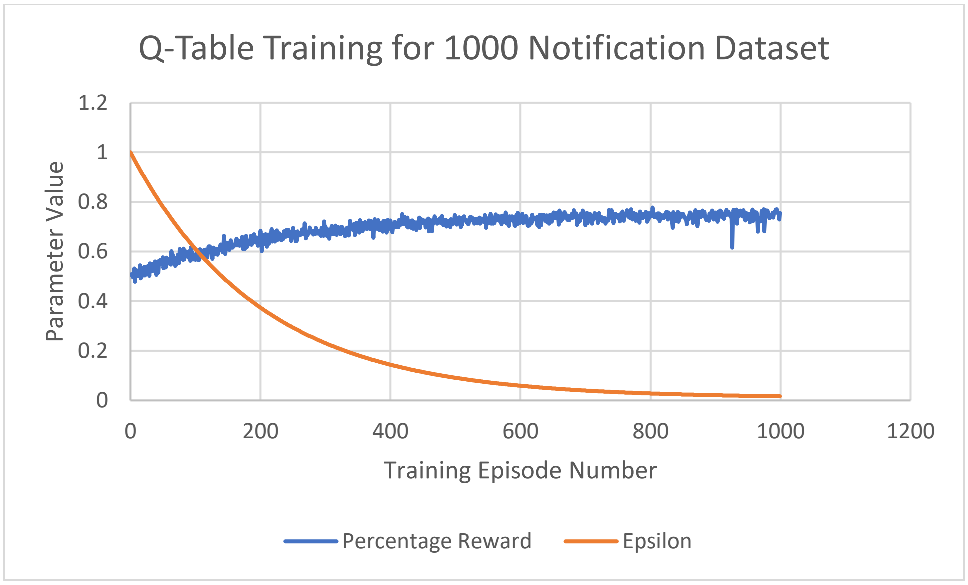

Figure 17: Q-Table training parameters for 1000 notification dataset (100 training notifications)

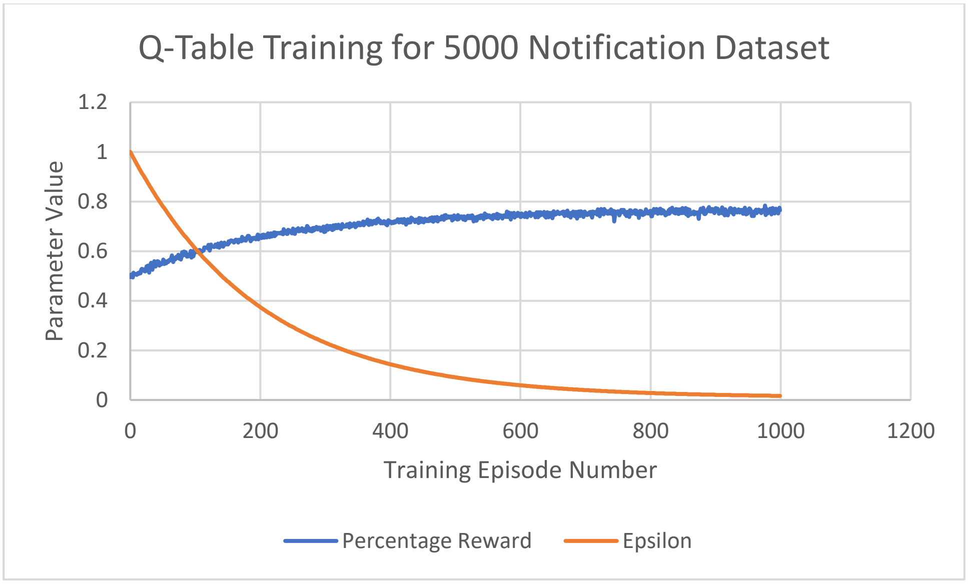

Figure 18: Q-Table training parameters for 5000 notification dataset (500 training notifications)

As can be seen by the training graphs above (Figure 15 to Figure 18), as the number of training notifications is increased, the variation in the training reward between training episodes decreases. This supports the previous theory that by using a larger training dataset, the results are more representative of the overall dataset.

5.1.3.2 Deep Q-Learning

The training metrics are shown below for the ten training episodes used by the Deep Q-Learning system.

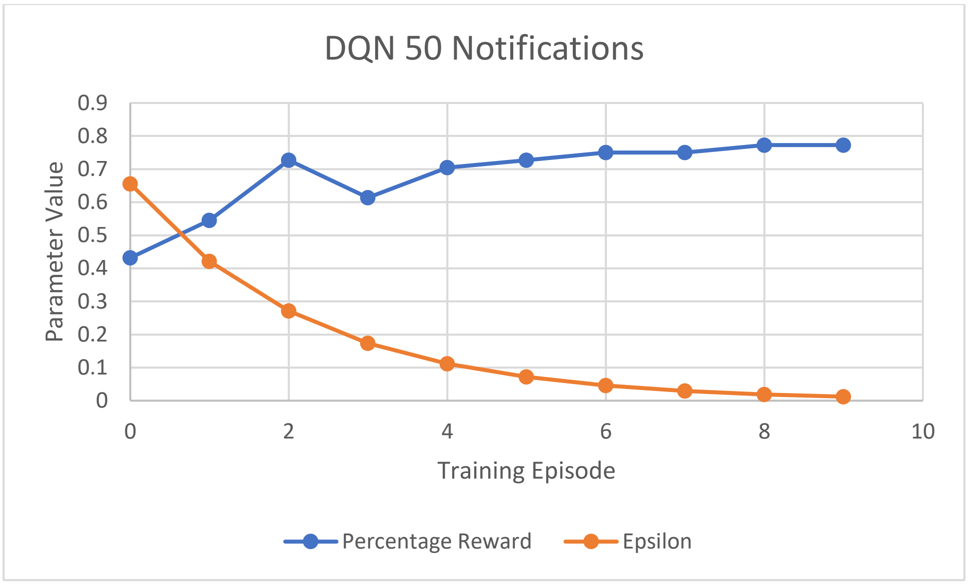

Figure 19: DQN training parameters for 50 notification dataset (5 training notifications)

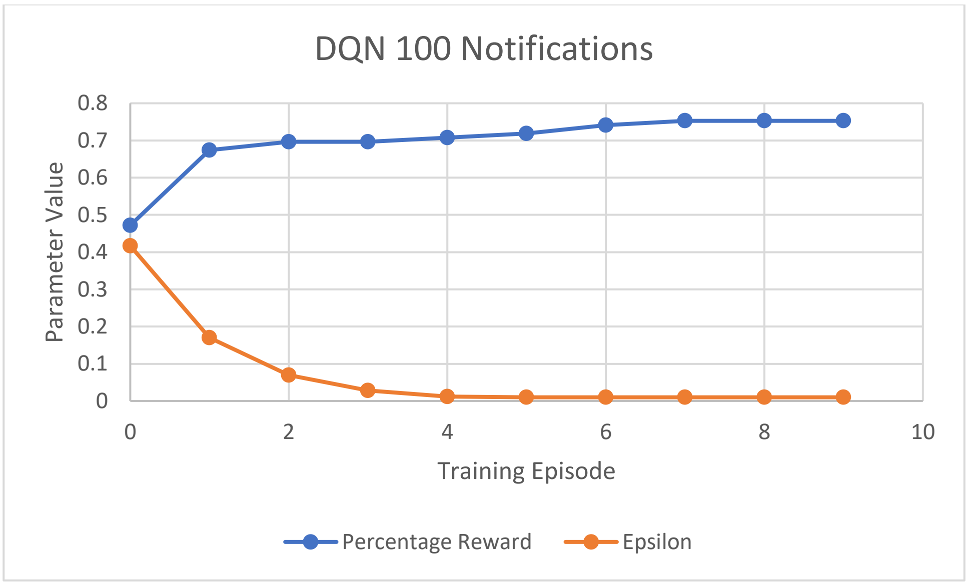

Figure 20: DQN training parameters for 100 notification dataset (10 training notifications)

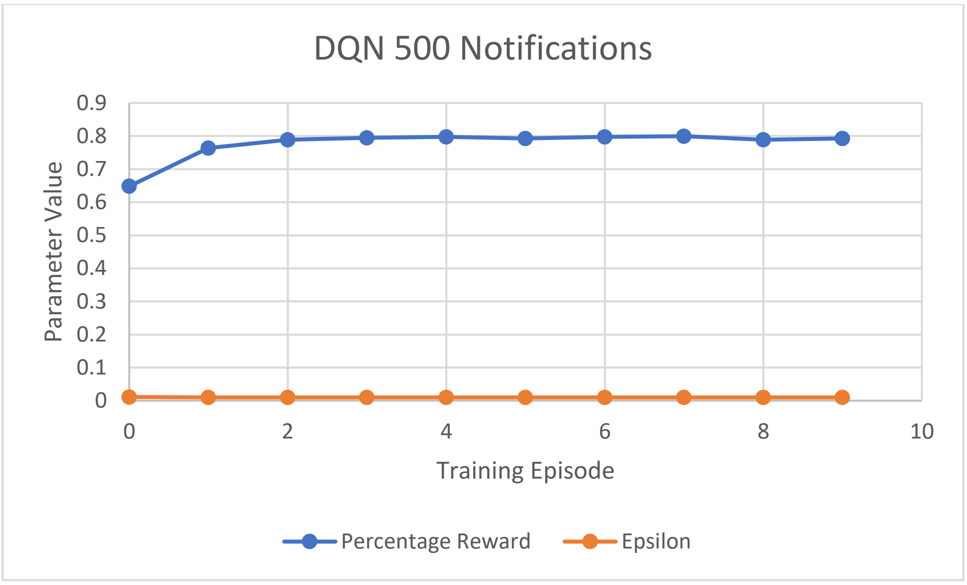

Figure 21: DQN training parameters for 500 notification dataset (50 training notifications)

What can be seen in the above training graphs for the DQN (Figure 19 to Figure 21) is that the epsilon value decays faster as the number of notifications in the dataset increases. This means that for large datasets, the training of the system is strongly predisposed to exploitation for later training episodes. The reason for this is that the epsilon decay value in this system was not correlated to the number of notifications used per episode, and as such when the number of notifications increased, the epsilon value decayed at the same rate per notification.

For future systems, the epsilon decay value should be associated with the overall number of system training weight update steps across the whole training process. It should not use a fixed value based off the total number of training episodes since the number of states to process per episode can change.

Despite this there is little difference in the value of percentage reward performance between the DQN trained on different dataset sizes.

5.1.4 Comparison of Systems

Both systems display similar performance in terms of the machine learning evaluation metrics for notification datasets between 500 and 2500 notifications. The DQN shows worse performance than the Q-Table system for very small datasets since the DQN was heavily biased towards negative results for small datasets. However slightly higher performance is shown by the DQN for the very large dataset of 5000 notifications.

The Q-Table showed training times around twice as fast as the DQN and implementation times around 40 times faster.

5.2 NUMBER OF NOTIFICATION FEATURES

The previous metrics measured for different notification sizes used the three features of app package, app category and time of day. To see if the number of features used to define the notification states had an impact on the system, different numbers of features were chosen for evaluation according to the table below (Table 2). An overall dataset size of 1000 notifications was chosen for each of the following feature spaces.

Table 2: Number of features used for notification states and their corresponding features

Number of Features	Feature Types	Number of Possible Notification States
4	app package, app category, time of day, and day of the week	2240
3	app package, app category and time of day	320
2	app package and app category	80
1	app package	16

5.2.1 Machine Learning Evaluation Metrics

Since the features have varying numbers of states, the following plots show the performance relative to the number of possible states (state space) for that number of features, instead of relative to the number of features

5.2.1.1 Q-Learning

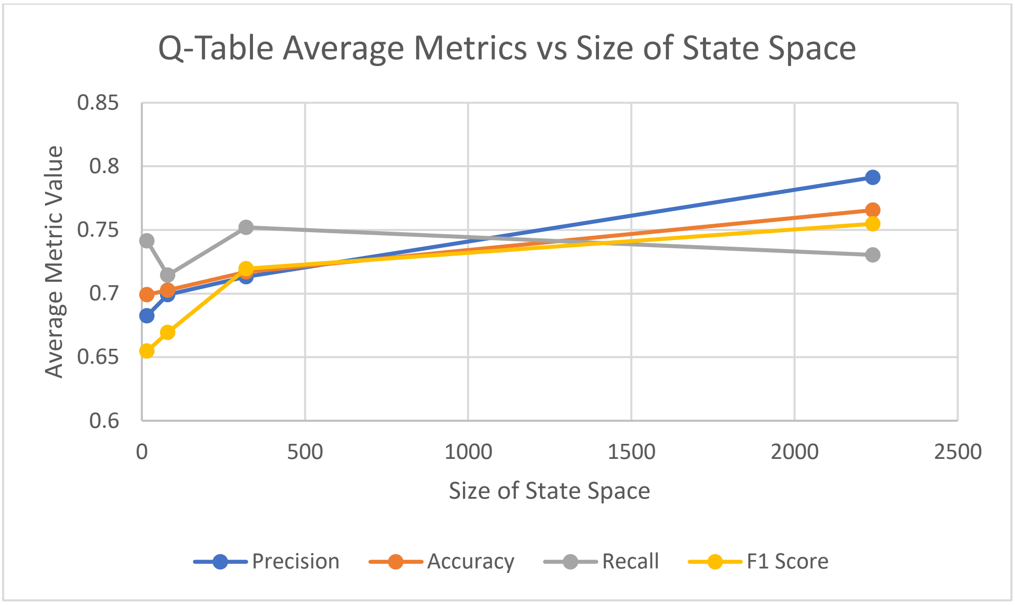

Figure 22: Average metrics across 10-fold cross-validation for the Q-Table implementation and their change with the number of possible notification states

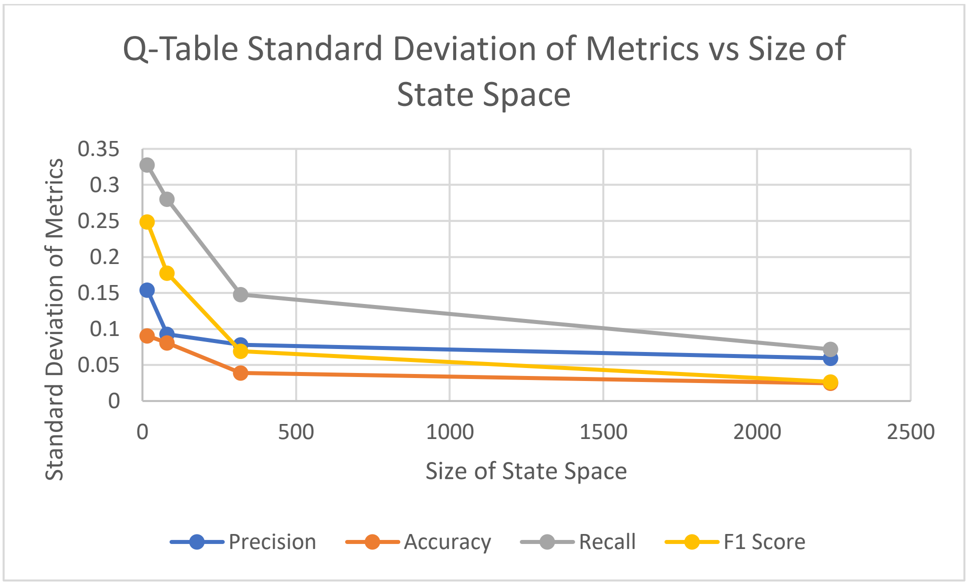

Figure 23: Standard deviation in results from Figure 22

5.2.1.2 Deep Q-Learning

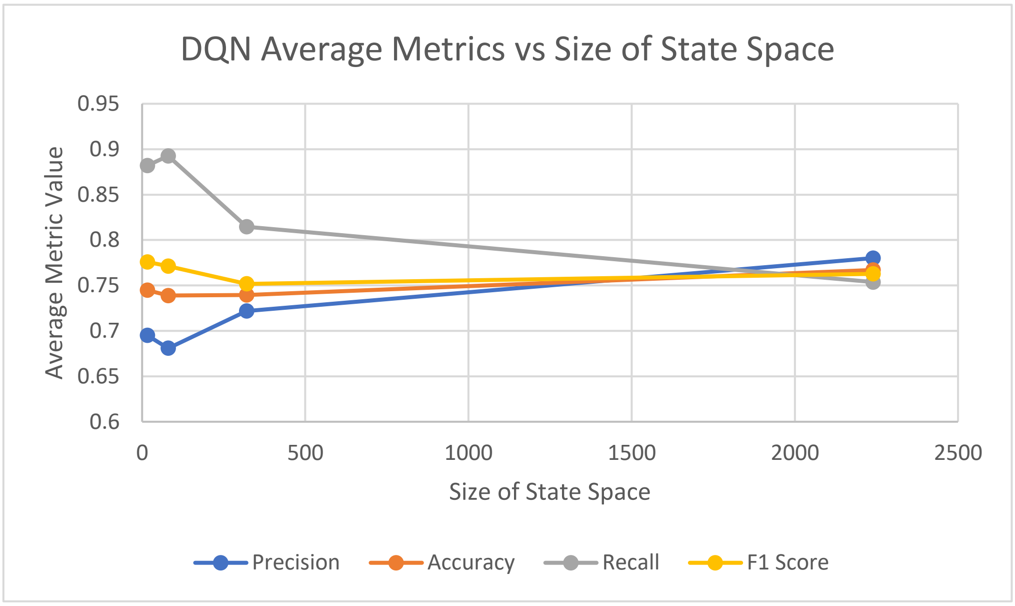

Figure 24: Average metrics across 10-fold cross-validation for the DQN implementation and their change with the number of possible notification states

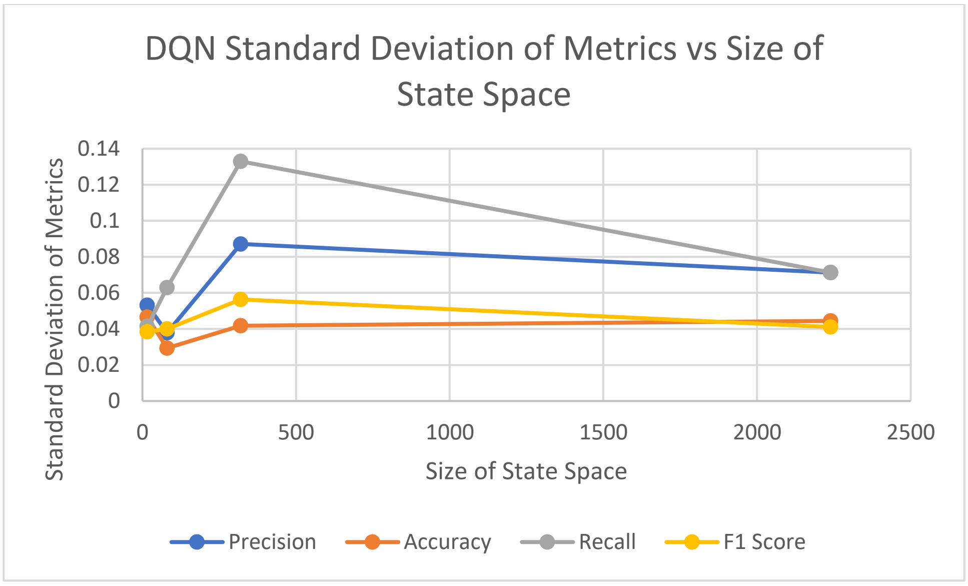

Figure 25: Standard Deviation of measurements in Figure 24

For the Q-Table implementation, there are improvements in all metrics except recall as the number of features increases. The standard deviation across k-steps also decreases as more features are used leading to more consistent performance of the system.

By contrast, for the DQN, the F1 Score value changes very little between different state space sizes. Instead there is a convergence of metric values as the state space grows 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 increases in size.

The standard deviation values for the DQN metrics were very low for all state spaces with values not exceeding 0.14 for any metric.

5.2.2 Computational Performance Metrics

5.2.2.1 Q-Learning

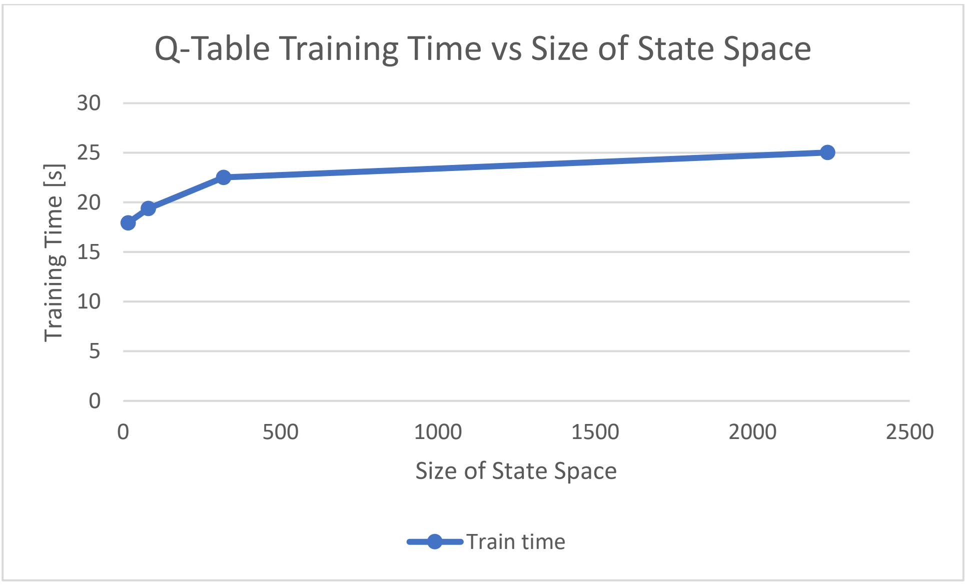

Figure 26: The effect of notification state space size on system training time

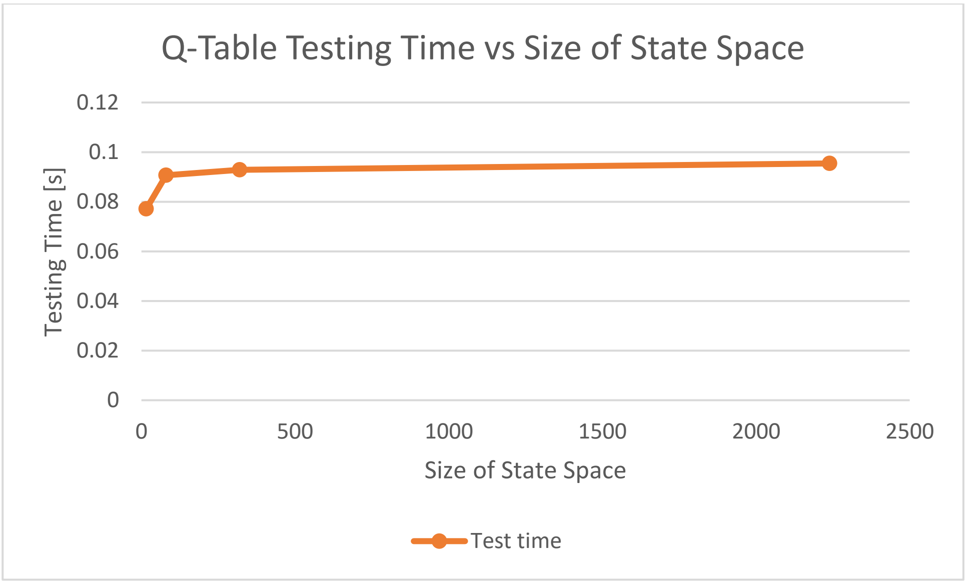

Figure 27: The effect of notification state space size on system testing time

5.2.2.2 Deep Q-Learning

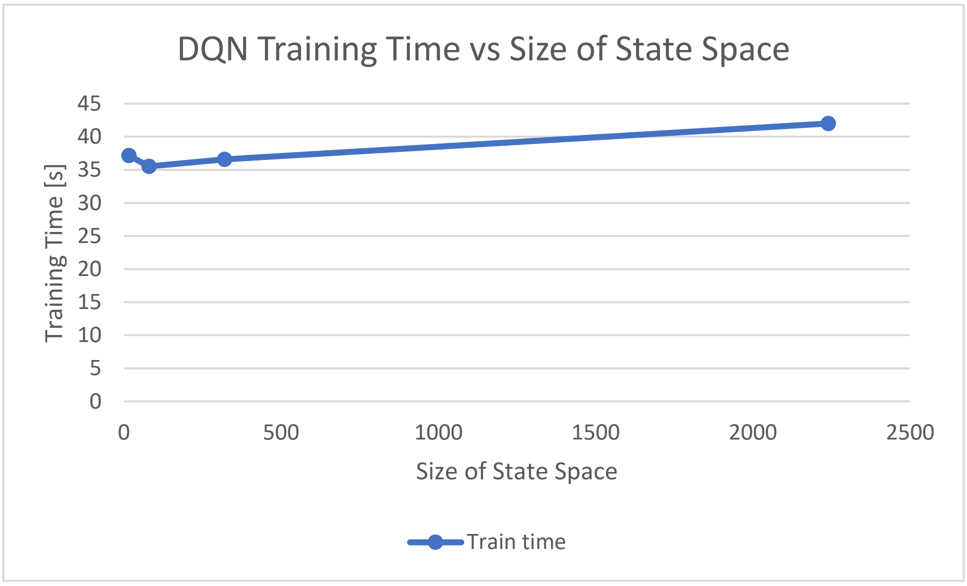

Figure 28: The effect of notification state space size on system training time

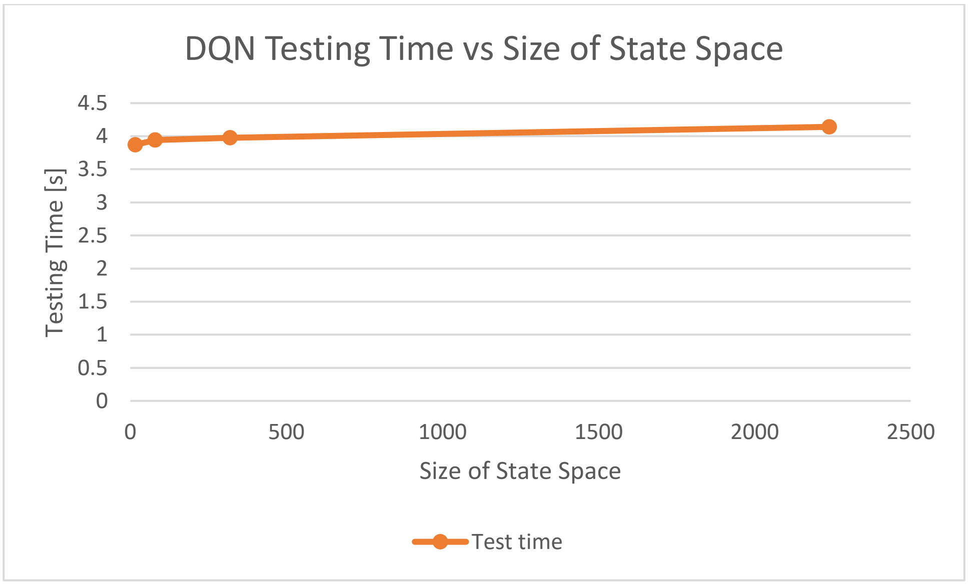

Figure 29: The effect of notification state space size on system testing time

For both systems there were small increases to the training times and very small increases to the testing times as more features were incorporated. An anomaly to this is the training time for a single feature space in the DQN system which was higher than training times for two and three feature spaces for the DQN. This could be explained by variations in the CPU load of the computer used to run these tests.

5.2.3 System Training Metrics

5.2.3.1 Q-Learning

The training metrics for one feature are near identical for those of two features so their graph has been omitted for brevity.

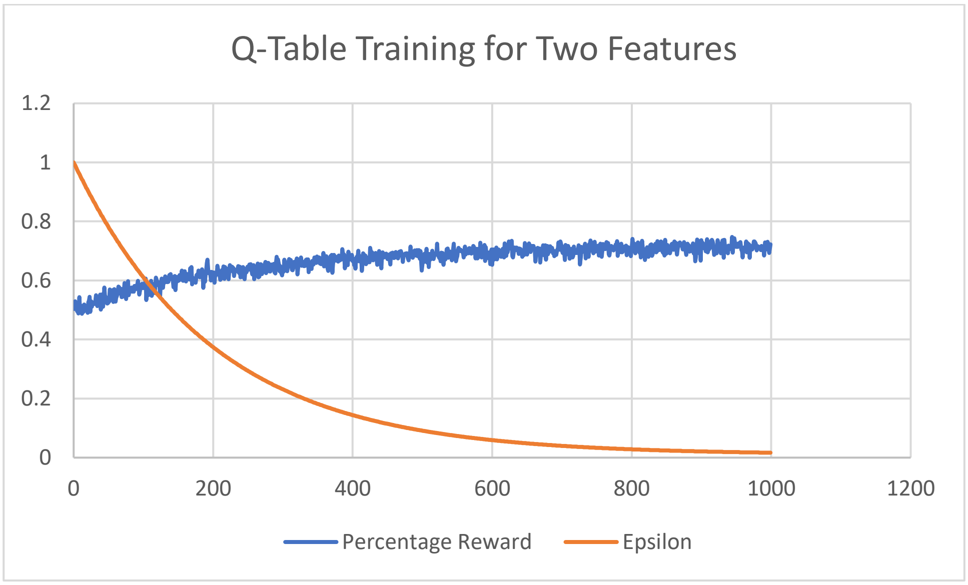

Figure 30: Q-Table training parameters using two notification features

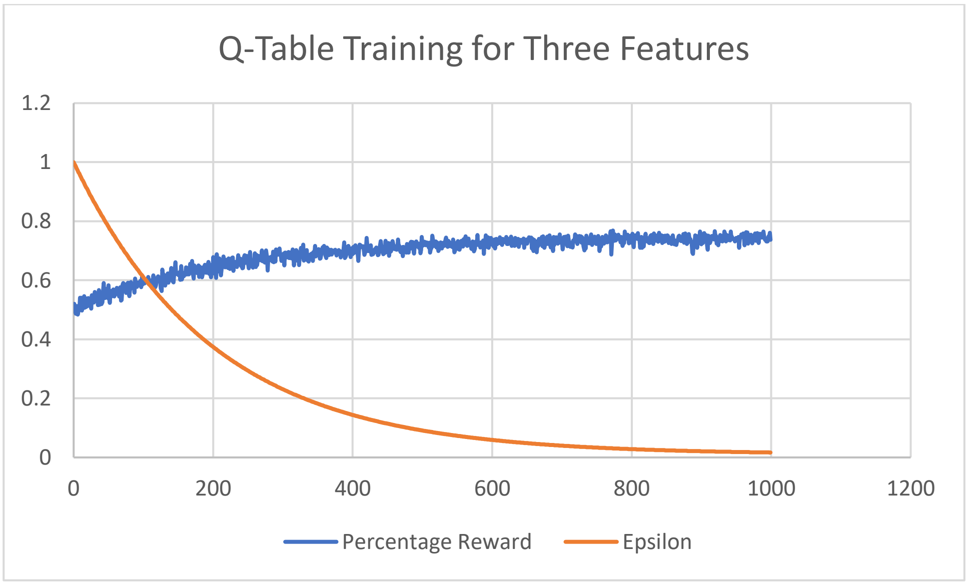

Figure 31: Q-Table training parameters using three notification features

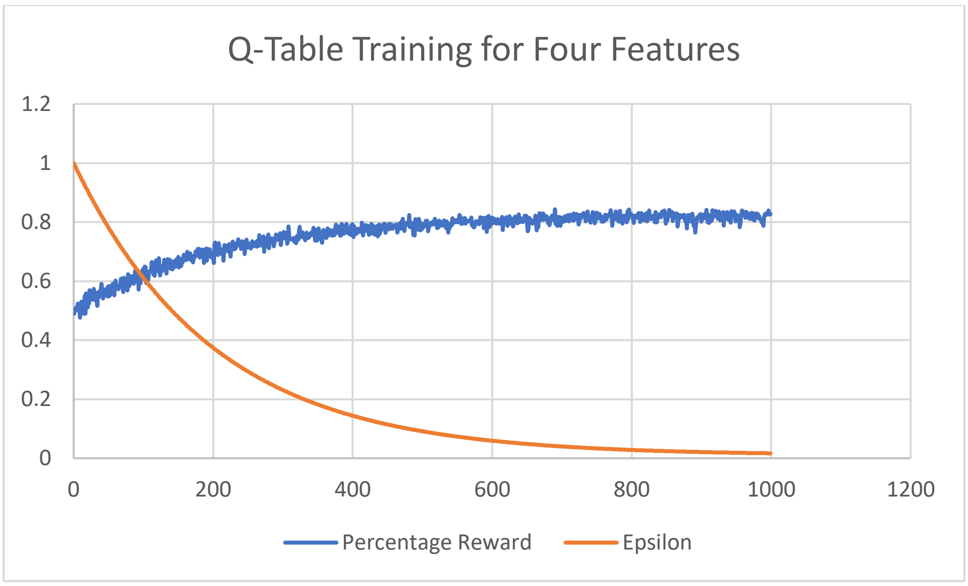

Figure 32: Q-Table training parameters using four notification features

There is an increase in the average percentage training reward for the last few episodes as the number of features increases. This increases from approximately 70% for one and two features to approximately 75% for three features and around 80% for four features.

5.2.3.2 Deep Q-Learning

There were few visible differences between the Deep Q-Learning training graphs for the different feature numbers since the epsilon value decayed quickly over the first two episodes. The percentage training reward increased similar to the Q-Table however, it increased from around 75% for one feature to around 81% for four features.

5.3 Single Feature Analysis

The systems were also analysed for their performance when only one feature was used to define the notification state space. Each of the four feature types (package, category, day of week, and time of day) were used alone and the system performance was measured for each. This was performed to identify which features were more effective for defining the notification state space to increase prediction performance. The Q-Table system with 1000 overall notifications was used for these measurements.

5.3.1 Machine Learning Evaluation Metrics

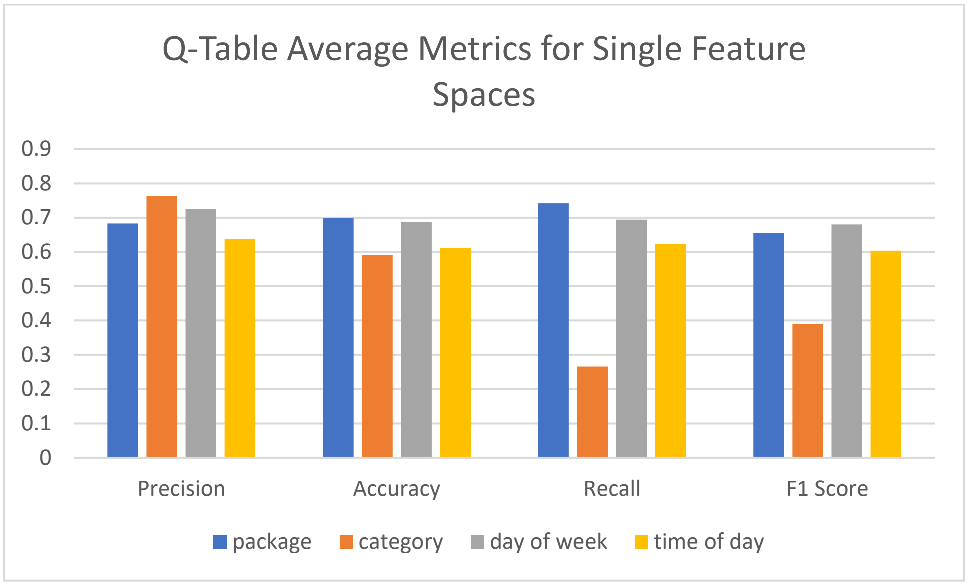

Figure 33: Average metrics across 10-fold cross-validation for the Q-Table implementation and their change with the choice of single feature space

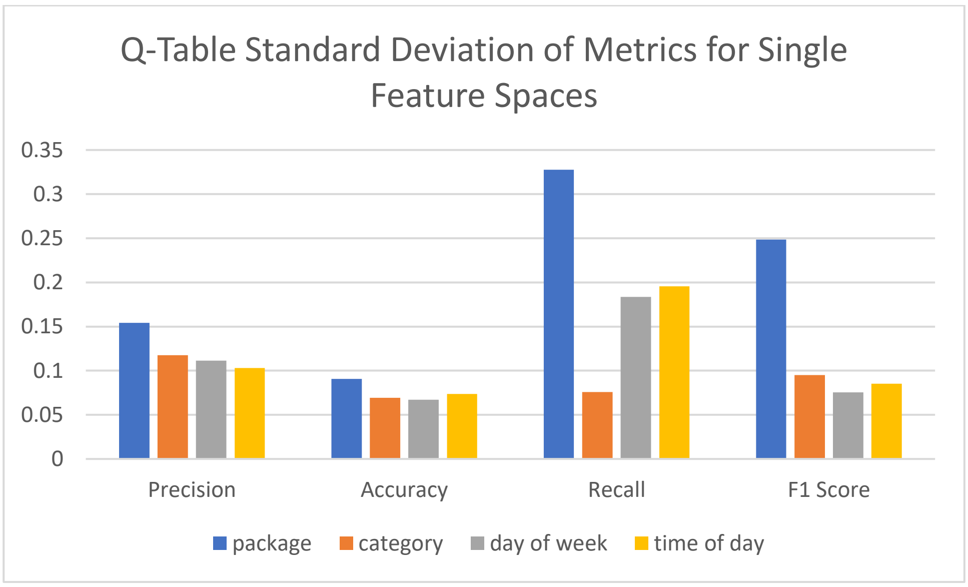

Figure 34: Standard deviation values for the measurements in Figure 33

From the single features analysed, the app package and day of week features showed the best performance with F1 Scores around 65% to 70%. The time of day feature showed worse performance with metrics around the 60% mark. Using only the category feature showed very poor behaviour an F1 Score below 40% and a recall below 30%. Similar to the results from small dataset sizes for the DQN, these low recall values indicate that the system was heavily biased towards guessing the negative action for each testing decision.

5.3.2 Computational Performance Metrics

Variations in testing and training time with changes in the number of features were very small and independent of the feature chosen.

5.3.3 System Training Metrics

All metrics for training were very similar to the ones previously investigated, with the exception of the category feature which is shown below (Figure 35).

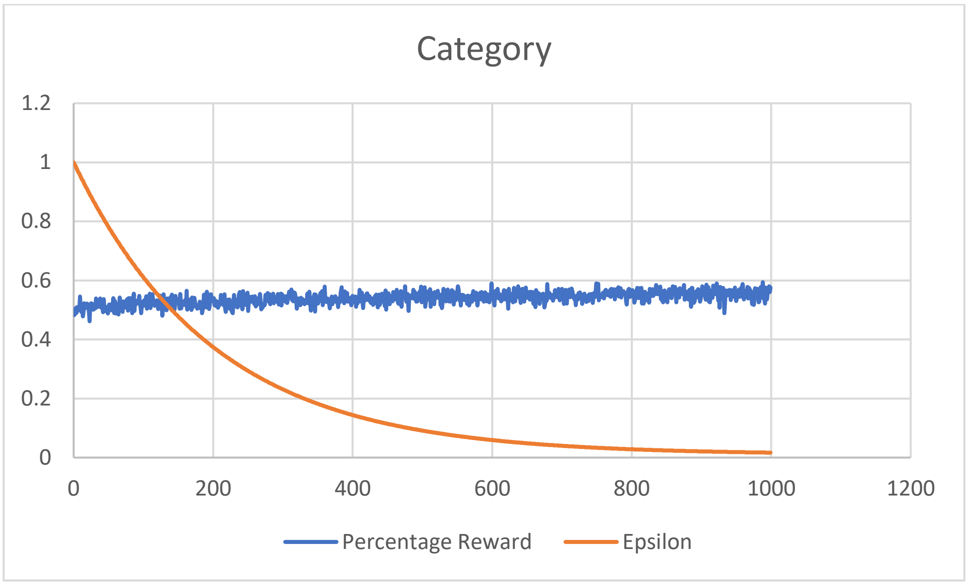

Figure 35: Training measurements when the only feature used is app category

The training metrics here show near random training performance when only the category feature is used. This can be explained by the category feature not imparting distinctive information for many notifications. From the 2500 notification dataset, 2065 of the notifications had the category "unknown", and 383 had the category "msg". Since around 80% of the notifications had insufficient data to distinguish between them, conclusions about notifications from the same category but with different interaction values could not be made.

6 Analysis and Discussion

6.1 System Parameters

6.1.1 DQN Performance Issues

Initially, training the DQN with 1000 training episodes and a replay batch size of 32 lead to very slow computation. The number of training episodes was reduced accordingly to 10 and performance was still far slower than the Q-Table equivalent. For example, a single k-step of the 50-notification dataset using only 10 training episodes required around 40 seconds to train. In an attempt to improve performance, the number of nodes in the neural network layers was reduced from 24 to 16, and later reduced to 8 and then 2. This reduction had little effect on the computation time. Afterwards, the number of nodes was returned to 24 and the batch size for the replay cache was changed instead. Changing the batch size from 32 notifications per new notification to 2 notifications, reduced the training time from around 40 seconds to 2-3 seconds, significantly improving the computation time. Reducing the batch size in this way did not seem to have any significant impact on the classification performance of the system. Since the replay cache is used once for each notification in the training set, replaying two of the past notifications for each new training notification seemed sufficient when compared to the performance trade-off of replaying 32 per new notification.

6.1.2 DQN Epsilon Rate and Dataset Size

As mentioned before for the DQN system, larger datasets reached low epsilon values in fewer episodes than smaller datasets due to the use of a static epsilon decay multiplier with no correlation to the overall number of replay iterations. This means that there was a lower proportion of exploration taken in larger datasets than smaller ones. For future implementations of this form of system, the epsilon decay value should be correlated to the size of dataset used.

6.1.3 System Learning Parameters

There is further potential for investigation into the impact that changing the system learning parameters such as epsilon decay rate, learning rate, and discount rate has on overall classification performance as these were not analysed in this report.

6.2 STATE AND ACTION SPACES

6.2.1 State Space

There is a trade-off in the choice of feature space size for reinforcement learning systems. If a large number of features to define each notification state is used, this results in a very sparse Q-Table where few new notifications are of a state matching one of the trained notification states. By comparison, a lower number of features means that new notifications are more likely to be encapsulated by previous training data, but edge case notifications are not properly accommodated. For example, a Q-Table has been trained to predict that a notification with time of day “morning” and category “unknown” should be interacted with. However, if a notification of this state is interacted with 70% of the time, the Q-Table system only using these two features will learn on average that the user will interact and predict incorrectly 30% of the time.

These probabilities are encapsulated in the different Q-values of the Q-Table however to make a binary decision on which action to take, the maximum argument is used. A DQN has similar issues surrounding this since the notification input is one-hot encoded for each state space.

There is a balance here where differences in notifications with similar feature values are accommodated, but the features are not too fine grain that uncommon notification states encountered in the future will encapsulated in existing trained data.

6.2.2 Action Space

The choice of user interaction as the action space was done to have a ground truth for comparing performance and validity of the systems’ predictions which also had a basis in the data generated for the synthetic dataset. In a real-world implementation of these systems, user interaction alone would not be used as the action space due to reasons previously mentioned in *3.1.6 Challenges in Acquiring Notification Data and Measuring User Interactions* with Notifications, such as reminder notifications which require delivery even though they are not interacted with. Instead an extended action space could be

6.3 COMPARISON TO STATE OF THE ART

6.3.1 System Analysis

The table below (Table 3) shows both the Q-Table and Deep Q-Network implementations compared to the other state of the art systems discussed in *3.3 State of the Art Systems*.

Table 3: State of the art systems with Q-Table and DQN systems

	Real User Data?	**Real Time Analysis?**	**Deployed on-device?**	**Type of ML used?**
A Context and User Aware Smart Notification System (Corno et al., 2015)	Synthetic information added to augment real user dataset	Separate training and classification stages. Training can take significant time depending on algorithm used.	No. Using Python script	Supervised (Support Vector Machine (SVM), Gaussian Naïve Bayes, and Decision Trees)
PrefMiner (Mehrotra et al., 2017)	Yes	Rules constructed when not in use. Rules implemented in real time.	Yes	Rule-based (with unsupervised learning for notification clustering)
Understanding and Managing Notifications (Pradhan et al., 2017)	Yes	No	No. Uses Weka, Matlab and vowpal-wabbit	Supervised (Random Forest, Decision Tree, SVM, Linear Regression)
C-3PO (Huang and Kao, 2019)	Yes	Yes	No. On remote server	Supervised (Deep Neural Network)
<u>Q-Table Implementation</u>	**No. Synthetic dataset from GAN**	**Could use real-time training and testing**	**No. Using Python script**	**Reinforcement Learning (Q-Table)**
<u>DQN Implementation</u>	**No. Synthetic dataset from GAN**	**Training times most likely too long for real-time CPU implementation**	**No. Using Python script**	**Reinforcement Learning (Deep Q-Network)**

6.3.2 Feature Analysis and User Interaction

When comparing the findings on which features were most important for classification, (Pradhan et al., 2017) found that "temporal features like hour of the day or temporally local event-based feature like last app use are the most important features" (Pradhan et al., 2017), whereas (Mehrotra et al., 2017) found that notification response, type and location were most important.

The findings from the *5.3 Single Feature Analysis* section indicated that the app package and day of week were the most performant features for classification, however the category feature used in this dataset contained very little useful data since most notifications were of category "unknown".

A benefit of the DQN and Q-Table implementations is that they require minimal user interaction when compared to previous systems such as PrefMiner which required direct user verification of rules.

6.4 Consequences of Using a Synthetic Notification Dataset

All analysis and evaluation of the Q-Table and DQN systems is based on the effectiveness of the synthetic notification dataset. The systems trained on the notification dataset showed the ability to determine useful classification features such as app package and time of day as well as demonstrating reasonable performance for action prediction with maximum performance for the metrics measured of approximately 75% to 80%.

A possibility for why the performance of the systems starts to level off after 500 notifications could be the simplicity of feature space. To preserve user privacy, the simplified feature space was used for synthetic notification generation and the maximum performance of the DQN and Q-Table systems may have been limited by this. As mentioned in *6.2.1 State Space* simplified feature spaces may not be able to fully encapsulate the reason why certain notifications were accepted and others were rejected, leading to notifications with the same state having different action values. This means that some of the notifications in the dataset could not be correctly classified without the use of additional features which impart information regarding user interaction.

Increasing the feature space size has a privacy trade-off however, since the inclusion of additional features in combination with others already available in the dataset increases the chance that privacy sensitive information from the original "in-the-wild" dataset could be gleaned.

Diversity of data within a synthetic notification dataset is important as well. It is possible that large synthetic datasets generated from a smaller sample of "in-the-wild" data could encapsulate all variation present in the original data, leading to reduced variation in notification states across the whole synthetic dataset. However for this synthetic dataset there were “31,239 notifications logged in the background service of the WeAreUs application” (Fraser, 2018) which provided a large enough “in-the-wild” dataset space when compared to the size of synthetic datasets used for evaluation in this project.

6.5 Computational Performance and Mobile Deployment

Results for computation time against state space size for the Q-Table implementation contradicted my initial assumption that there would be a liner relationship between the two. This assumption was made on the basis that as the Q-Table grew in size, there would be a corresponding computation time trade-off. Instead there was a minor increase in computation time from 18 seconds for 16 states to 25 seconds for 2240 states.

This in combination with the results for notification dataset size against computation time support the idea that the Q-Table size was less important than the number of update and query function calls.

An important point to note is that both systems were trained on CPU architecture. Neural network systems display higher performance when deployed on GPU architecture and for mobile specific deployment, TensorFlow Lite (Google LLC, 2019b) is currently available and can implement deep neural networks on mobile devices.

A Q-Table system could be deployed using any conventional programming language for mobile applications. There is the potential for future research to be conducted into the performance effectiveness of both DQN and Q-Table systems when deployed on mobile devices.

6.6 Change in User Context

When reacting to a change in user context, the speed at which the two systems could react is based on a few parameters. For the Q-Table system, it would depend on the size of the training notification dataset relative to the size of the new context notification set. For the DQN, the size of the replay cache would be the main factor in determining adaptability to context change. A larger cache would cause the system to change more slowly, but also incorporate notification data from further in the past, whereas a smaller replay cache would respond more quickly to recent changes.

With this in mind, the DQN system with an appropriately chosen replay cache size would be more adaptable than the corresponding Q-Learning system.

6.7 System Comparisons

Both the Q-Table and DQN implementations have application depending on the use case. For systems which can use very large datasets, the DQN system would be preferred as shown by its improved classification metrics for the 5000 notification dataset (precision 73.9%, accuracy 79.1%, recall 90.0%, F1 Score 81.0%) when compared to the Q-Table (precision 72.6%, accuracy 76.5%, recall 85.8%, F1 Score 78.2%). If the DQN system is also able to use a GPU to increase performance, further potential for improvement over the results shown in this project.

If the size of the dataset is rather small in the order of 500 notifications or less, neither system is ideal although the Q-Table system shows significantly better performance than the DQN system. A Q-Table system would also be used if a real-time training and implementation was desired, or if there were constraints on the computation time for the implementation since the Q-Table implementation is 40 times faster than the DQN implementation on CPU.

If a very large feature space is used there may be memory constraints on the maximum size of Q-Table that could be created. A DQN would avoid these problems by having a relatively fixed memory requirement.

7 CONCLUSION

7.1 OVERVIEW

This project has highlighted the issues surrounding mobile notification managements and the call for NMSs to reduce their negative impacts. Four systems were analysed on their implementation and effectiveness as well as the main findings from their respective papers. From this, gaps in research were found in the areas of reinforcement learning for NMSs and the use of synthetic notification datasets for their training and evaluation. Two reinforcement learning systems implementing Q-Learning and Deep Q-Learning were then developed and evaluated using a synthetic notification dataset.

This paper demonstrated how the development of a reinforcement learning mobile NMSs could be conducted in a privacy sensitive manner with consideration for the privacy of notification dataset contributors.

7.2 MAIN FINDINGS

The Q-Learning and Deep Q-Learning systems implemented were found to be effective with maximum performance in machine learning metrics of precision, accuracy, recall and F1 Score of approximately 80%. While higher performance would be required for the full implementation of a system such as this to avoid interruption of important notifications, further performance gains could be obtained from optimization of the reinforcement learning system parameters. Additional notification features could also be used, although using additional features may encompass a corresponding trade-off in user privacy.

The project overall found that reinforcement learning algorithms could be implemented effectively for mobile NMSs and were found to be performant when evaluated against synthetic notification data.

7.3 WEAKNESSES AND LIMITATIONS

All results are based on the effectiveness of the synthetic dataset used and any biases or inconsistencies within this dataset would reflect in the results. The synthetic dataset also implements simplified feature spaces which may have reduced the effectiveness of classification as mentioned in *6.4 Consequences of Using a Synthetic Notification Dataset*.

In terms of performance testing, the results measured for both systems are highly dependent on the hardware used for those tests. Since the systems were not deployed on mobile devices, the assumption being made is that high performance in desktop computer environments will translate to similar performance on mobile devices. The system parameters for each system were also not strongly optimized and further performance improvements between the systems could be obtained.

If a DQN system were to be developed for use by mobile users, it would definitely use some form of GPU implementation. The DQN system for this project was trained on CPU to provide direct parallels with the Q-Table implementation and because performance results on a desktop GPU would be heavily dependent on the GPU device used. The only way to obtain effective understanding of the DQN's performance on mobile GPUs would be to deploy the system on mobile.

7.4 Recommendations

From these results, the recommended system to implement is heavily dependent on the use case. The Q-Table implementation would be recommended for systems with small to medium datasets, a computation time requirement or the need for real-time implementation. The DQN implementation by contrast would be better for large datasets and large feature spaces and could be further improved if significant leverage of GPU devices is possible. The DQN also has a relatively fixed memory requirement when compared to the O(n) scaling of the Q-Table size with number of notification states, and so would be recommended for systems with memory constraints.

These use of synthetic notification datasets is also recommended since they demonstrated the ability to determine performant features and compare the performance of different NMSs for a variety of dataset parameters. Using such systems could be useful for NMS design before they are implemented to assuage privacy concerns surrounding the collection and storage of "in-the-wild" notification data for system design.

7.5 Further Research

There is potential for further research in the parameter optimization of both the DQN and Q-Table systems to improve classification and computation performance further as the system parameters were not heavily optimized in this project.

There is also the area of mobile deployment which requires further investigation for both systems. The Q-Table system could be implemented via the mobile platform’s corresponding programming language whereas the DQN system could be implemented through Tensorflow Lite (Google LLC, 2019b) for the neural network component of the system.

An important area of research would be to compare the performance of reinforcement learning systems trained on “in-the-wild” data to those trained on a synthetic notification dataset with comparable features. If strong correlations between the performance of systems trained on synthetic data with their performance using “in-the-wild” data can be made, it would further reinforce the effectiveness of synthetic datasets for mobile NMS system design and evaluation.

8 REFERENCES

ASHRAF, M. 2018. *Reinforcement Learning Demystified: A Gentle Introduction* [Online]. Available: https://towardsdatascience.com/reinforcement-learning-demystified-36c39c11ec14 [Accessed 13 March 2019].

BAILEY, B. P., KONSTAN, J. A. & CARLIS, J. V. The Effects of Interruptions on Task Performance, Annoyance, and Anxiety in the User Interface. 2000. 757 - 762

CORNO, F., DE RUSSIS, L. & MONTANARO, T. A context and user aware smart notification system. Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on, 2015. IEEE, 645-651.

FELT, A. P., EGELMAN, S. & WAGNER, D. 2012. I've got 99 problems, but vibration ain't one: a survey of smartphone users' concerns. *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices.* Raleigh, North Carolina, USA: ACM.

FRASER, K. 2018. Toward Coaching Improved Smartphone Notification-Engagement Behaviour∗. *Mobiquitous'18.* New York, USA.

FRASER, K. 2019. *WeAreUs* [Online]. Available: https://www.weareus.eu/ [Accessed 23 February 2019].

FRASER, K., YOUSUF, B. & CONLAN, O. Synthesis & Evaluation Of A Mobile Notification Dataset. Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization., 2017. ACM, 179–184.

GOOGLE LLC. 2019a. *NotificationListenerService | Android Developers* [Online]. Available: https://developer.android.com/reference/android/service/notification/NotificationListenerService [Accessed 28 February 2019].

GOOGLE LLC. 2019b. *TensorFlow Lite | TensorFlow* [Online]. Available: https://www.tensorflow.org/lite/ [Accessed April 13 2019].

GRAZ UNIVERSITY OF TECHNOLOGY. 2006. *5-Fold Cross-Validation* [Online]. Available: http://genome.tugraz.at/proclassify/help/pages/XV.html [Accessed 13 March 2019].

HUANG, T. H.-D. & KAO, H.-Y. J. S. C. 2019. C-3PO: Click-sequence-aware deeP neural network (DNN)-based Pop-uPs recOmmendation.

IQBAL, S. T. & BAILEY, B. P. 2010. Oasis: A framework for linking notification delivery to the perceptual structure of goal-directed tasks %J ACM Trans. Comput.-Hum. Interact. 17**,** 1-28.

JAMES, G., WITTEN, D., HASTIE, T. & TIBSHIRANI, R. 2013. *An Introduction to Statistical Learning with Applications in R*, Springer.

LEIVA, L., B, M., #246, HMER, GEHRING, S., KR, A., #252 & GER 2012. Back to the app: the costs of mobile application interruptions. *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services.* San Francisco, California, USA: ACM.

MEHROTRA, A., HENDLEY, R. & MUSOLESI, M. 2017. Interpretable Machine Learning for Mobile Notification Management: An Overview of PrefMiner %J GetMobile: Mobile Comp. and Comm. 21**,** 35-38.

MEHROTRA, A., PEJOVIC, V., VERMEULEN, J., HENDLEY, R. & MUSOLESI, M. My phone and me: understanding people's receptivity to mobile notifications. Proceedings of the 2016 CHI conference on human factors in computing systems, 2016. ACM, 1021-1032.

OPENAI. 2019. *A toolkit for developing and comparing reinforcement learning algorithms* [Online]. Available: https://gym.openai.com/ [Accessed 11 April 2019].

OULASVIRTA, A., RATTENBURY, T., MA, L. & RAITA, E. 2012. Habits make smartphone use more pervasive %J Personal Ubiquitous Comput. 16**,** 105-114.

PIELOT, M., CHURCH, K. & OLIVEIRA, R. D. 2014. An in-situ study of mobile phone notifications. *Proceedings of the 16th international conference on Human-computer interaction with mobile devices &#38; services.* Toronto, ON, Canada: ACM.

PRADHAN, S., QIU, L., PARATE, A. & KIM, K.-H. Understanding and managing notifications. INFOCOM 2017-IEEE Conference on Computer Communications, IEEE, 2017. IEEE, 1-9.

SALLOUM, Z. 2018. *Monte Carlo in Reinforcement Learning, the Easy Way* [Online]. Available: https://medium.com/@zsalloum/monte-carlo-in-reinforcement-learning-the-easy-way-564c53010511 [Accessed 11 April 2019].

SHIRAZI, A. S., HENZE, N., DINGLER, T., PIELOT, M., WEBER, D. & SCHMIDT, A. 2014. Large-scale assessment of mobile notifications. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* Toronto, Ontario, Canada: ACM.

SIMONINI, T. 2018a. *Adding 'Deep' to Q-Learning* [Online]. Available: https://cdn-images-1.medium.com/max/1600/1*w5GuxedZ9ivRYqM_MLUxOQ.png [Accessed 31 March 2019].

SIMONINI, T. 2018b. *Bellman Equation* [Online]. Available: https://cdn-images-1.medium.com/max/1200/1*jmcVWHHbzCxDc-irBy9JTw.png [Accessed 12 April 2019].

SUTTON, R. 2019a. *OpenAI custom Gym environment for mobile notification reinforcement learning* [Online]. Available: https://github.com/suttonr0/gym-notif [Accessed 11 April 2019].

SUTTON, R. 2019b. *Reinforcement Learning Q-Table and DQN agent for use with gym-notif Gym environment* [Online]. Available: https://github.com/suttonr0/RL-QTable [Accessed 11 April 2019].

YANG, Z., YANG, M., ZHANG, Y., GU, G., NING, P. & WANG, X. S. 2013. AppIntent: analyzing sensitive data transmission in android for privacy leakage detection. *Proceedings of the 2013 ACM SIGSAC conference on Computer &#38; communications security.* Berlin, Germany: ACM.