Project 3: Report

Managing Smartphone Interruptions through a Smart Notification System

Vivek Paul Joseph
156330004
Interaction Design
M.Des (2015-17)

Guide: **Prof. Jayesh Pillai**

IDC School of Design
Indian Institute of Technology Bombay

Aknowledgements

I would like to thank Prof. Jayesh Pillai for his support and guidance through the duration of this project. I am grateful to Priof. Anirudha Joshi, Prof. Girish Dalvi, Prof. Ravi Poovaiah and Prof. Venkatesh Rajamanickam at IDC for their valuable feedback and suggestions.

Table of Contents

1. Introduction

Notifications could be a major source of disruptions in smartphone users' activities. While some of these notifications are important and useful, others are either unimportant or not urgent. These notifications can potentially break the flow of the activity that the user was engaged in. For certain activities and for certain users, it may not be of much consequence. But in some cases, the time taken back to regain the focus in the activity may be longer; and hence could affect their overall productivity. And especially in a case where the notification was regarding something unimportant, the cost (break in flow) does not justify the benefit (the information from the notification). While the user can simply switch off the phone, it may not necessarily be the best solution since they may miss an important call or message. Another option would be to mute notifications from specific apps for a certain duration. But such control requires the user to take time to set them up. Such a solution requires active control from the user. This could work for some users, but not all. Currently the android OS (5.0 and up) and certain apps allow muting notifications from specific apps. But that may not be the ideal solution since the user may not want only certain notifications from the app (for example: Muting the messaging app to get rid of advertisement messages would also block important messages from people or banks).

While there are lots of things for people to get distracted by, smartphone notifications are one of them; and controlling the unwanted notifications would make it one less source of distraction. One possible method to achieve this could be using a notification control system that learns user preferences based on their usage patterns and automatically prioritises these interruptions, sorts them and presents the user with only the important ones. This would require only minimum active control from the user and hence would work for a wider range of users (compared to solutions requiring active control) and still provide a personalised experience to the user. This project is an attempt to develop the framework for such a system.

2. Research Problem

Notifications popping up on smartphones while user is engaged in some activity could potentially cause a temporary disruption in their flow. Some of these notifications (for example, advertisements, unimportant mails, etc.) are not urgent and/or don't require immediate attention. In such cases, the interruption is not worth the user's time or disruption in their work flow.

2.1 Opportunity for Design Intervention

A personalised notification system that prioritises the notifications based on the user's behaviour (and interrupt the user only when necessary) could reduce the number of unwanted disruptions and provide the user a relatively continuous work flow. Such a system would interrupt the user only when it is important to the user.

This system would need interactions for the user to communicate the receptiveness towards a notification. The presentation of the notifications based on their priority could also be explored to provide the user a better experience with their smartphones.

2.2 Approach

Broadly, the approach can be split to :

1. User Behaviour Study
2. Development of a Notification Prioritisation System
3. Evaluation of System Performance

2.2.1 User Behaviour Study:

This stage covers the secondary and primary research. Secondary research being the existing research material; and primary research being done through interviews and data collection. For the data collection part, a monitoring app was developed and installed in the users' smartphones. This app monitored a wide range of variables on the user's phone and logged them.

2.2.2 Development of a Notification Prioritisation System:

The insights and consequent design implications that were drawn from the secondary and primary research were used to develop the UI and UX elements of the system.

The data collected for each user were, then analysed to identify patterns that the prioritisation system can be built upon. This step involved studying the collected data to determine the potential factors that can be used as indicators towards the priority that needs to be set to a particular type of notification. Machine Learning algorithms is one of the methods that was explored for the analysis of the collected data.

Each user's data was analysed separately in order to get personalised patterns. This stage would also involve the design of the UI and UX elements of the aforementioned system, where different methods of representation of the notifications were explored to come up with one that works for this system.

2.2.3 Deployment/Evaluation of System Performance

While ideally this system should be part of the smartphone OS itself, for the scope of this project, an application can be developed incorporating the findings from the previous step. This application would simulate how to:

- manage which notifications get shown to the user;
- in what way it will be presented to the user; and
- how the user would interact with the notifications.

2.3 Deliverables

A framework for a notification prioritization system that could demonstrate the concept pitched in the project. This would include:

- Identified factors that could potentially affect the user's behaviour towards a notification
- A prioritization algorithm based on the identified factors
- A semi-functioning prototype

Since Android doesn't support adding of new interactions to the notification bar, the UI and the functioning of the system have to be prototyped separately. This would act as a proof of concept of the system. The prototype would have two parts:

- A framework which collects user's notification data, trains an algorithm with the data and can identify low-priority notifications
- A UI prototype that demonstrates the working of the interactions

3. Secondary Research

In this section, existing research material on the relationship between users and notifications are discussed. The aim here was to look for what the users wanted from the notifications and what bothered them on the qualitative front; and what kind of notifications they got, how they responded to the notifications and other related factors of interest on the quantitative front. Research papers on how to recognize disruptive notifications were also studied for methods that could be used to identify and manage such interruptions.

This stage included the study of online material on machine learning algorithms, their use, applicability and limitations with respect to developing algorithms that could identify disruptive notifications. Android OS ecosystem was also studied in order to ensure that the design choices that get made fit within the ecosystem.

In the subsections, the takeaways and points of interest that are applicable to the project are discussed.

3.1 Existing Research on User Interaction with Notifications

An ethnographic study on 'Mobile Notifications and attention management' by a team from Google [1] indicated that some users were attached to their phones out of fear of missing something important. They found that even the participants who blocked out all notifications were still frequently checking their phones manually. They also identified messages (from people) and task notifications as the ones that were considered important by most participants.

Another in-situ study by M. Pielot et. al [2] that involved 15 participants, found an average of 63.5 notifications per user per day. Most of these notifications were from messengers and emails. This study included a quantitative and subjective study of the notifications that the users received and how quickly they responded to them. They found that the participants reported increasingly negative emotions with the increase in number of email and social network notifications received over the course of the day. The study indicated that the fastest attended notifications were those generated by messengers and social network applications. An interesting finding from this study was that there was no evidence of slower response to notifications when the phone was on silent mode. This would indicate that the users are almost as likely to get interrupted with their phone in silent mode as they would be otherwise. While some users took the explicit decision to delay responding to certain notifications (eg: messenger notifications), there was always an underlying social pressure to respond asap.

3.2 Existing Research on Interruption Management for Smartphones

Previous research in the area of smartphone based interruption management have relied on the use of machine learning (ML) algorithms to predict whether a given interruption is relevant to the user or not [3,4,5,6].

InterruptMe[6] focuses on the interruptibility of the user to predict opportune moments for sending notifications. The main goal of this experiment was to identify appropriate moments to interrupt the user. Their experiment consisted of an app that is installed on the users' phones that would generate notifications periodically. The user, then, would respond to their interruptibility through explicit input methods (Fig. 1). This data, (along with the metadata for each notification) was used to train an interruptibility model that is personalized to the user. While such a model would be very useful in minimizing interruptions and for better overall user experience, such an approach would work only in an experimental setting, since most users would not want to spend time and effort training the system.

J Smith et. al [4] explores different machine learning algorithms to identify disruptive phone calls. Here, the data collection to fuel the ground truth data was done through an app which tracks the response of the user to a particular call, i.e., whether the user answers, declines or puts it on silent mode. This approach for collecting the user response would work specifically only for calls since the user HAS to respond to the call. The same is not the case with notifications, where the user responses can't be directly correlated to the relevance of the notification.

The primary research in case of InterruptMe system consisted of periodically collecting explicit input on their interruptibility from the user using a likert scale; and using this dataset to train the ML algorithm to predict opportune moments in the future.

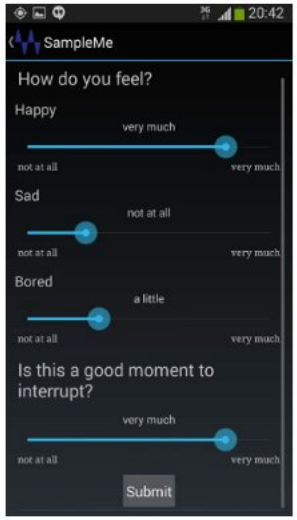


Fig. 1: InterruptMe[2] interface

Other similar projects like RingLearn system[7] and In-Context system [8] learn when to silence phone call notifications based on different variables. But while phone calls are a source of interruptions that a smartphone affords, it is not the only one. Notifications are a major source of interruptions.

4. Primary Research

The primary research consisted of mainly two steps:

1. Preliminary interviews with smartphone users

2. Collection of user behaviour factors and preferences; and factors that can be used to determine patterns that can be used to predict the relevance of a notification to the user.

4.1 Preliminary Interviews

In this stage, the attempt was to collect qualitative data on how users interact with notifications in their current state. Semi structured interviews were conducted with 23 android smartphone users. The main headers that the interviews touched upon were:

- How many notifications the users think they get on average per day?
- Do they find notifications disruptive?
- Methods that they use to manage interruptions
- Why they don't use interruption management methods that they don't use

Most users reported that they received 'a lot' of notifications everyday. Understandably, they weren't keeping track of the exact number. But while most of the users found some notifications to be bothersome, there were a couple of users who were 'used to' them and weren't bothered much by them. But all of the users agreed that they would like some of the notifications to stop.

Regarding the management of notifications, out of the 23 interviewees, 4 used the 'block app' option within android to prevent specific apps from sending messages. Out of these 4, 1 user blocked individual contacts (mostly advertisements) from sending messages. It was interesting to note that 22 of the interviewees used the 'mute' option in WhatsApp. The muted conversations were all group chats. They felt no urgency to respond to or read the group chats, unlike the individual chats, where they felt the need to respond. This observation was inline with the previous studies (discussed in section 3.1) in the same field.

4.2 User Phone Data Collection

The aim of this step is to collect notification data from the users' smartphones. This data can then be analyzed and used for developing an algorithm that can identify a disruptive notification. It needs to be ensured that all of the variables that are tracked can be categorized into a finite number of categories so that it can be fed into the machine learning algorithm. The parameters were selected with the intent of measurement/tracking on phones running on android platform.

4.2.1 Selection of Parameters to track

The parameter selection was done based on the different variables that can be measured and tracked by a smartphone. The preliminary list of parameters is listed below:

- App/Package Name : The app that is throwing the notification
- Date of receipt : This is to be then converted into weekday/weekend while feeding to the machine learning algorithm
- Time of Receipt
- Time of Consumption: When the user opens/clears the notification
- Priority of message : Integer values ranging from -2 to 2 that are assigned by android system/application
- Text Content of the notification : To be analyzed later for better classification
- Title of notification
- User Activity at the point of receipt of notification
- User Activity at the point of consumption of notification
- User state at the point of receipt of notification (If the user is engaged in any activity based on their calendar status)
- Was the user using the phone at the point of receipt of notification
- User Location on notification receipt
- User Location on notification consumption
- User response as to whether they find the notification relevant

At the point of working on the collection of these variables, it was observed that some of the variables were harder to track than others mainly due to the fact that they were too resource/battery consuming. Based on this observation, a few of the parameters were dropped, namely:

- User Activity at the point of receipt of notification
- User Activity at the point of consumption of notification
- User state at the point of receipt of notification (If the user is engaged in any activity based on their calendar status)
- User Location on notification receipt
- User Location on notification consumption

While this would potentially make the predictions a bit less accurate, till a point when these variables can be recorded with less consumption of resources, they were dropped. At such a point, they can be added as features into the machine learning algorithm to improve it.

4.2.2 Data Collection Tool

The Data Collection Tool records the parameters identified in Section 4.1 and provides the users with a method to record their response to any notification. While most of the parameters that are recorded are automatic, the user's response (about the relevance of the notification) is gathered from the users themselves. This is done by showing a notification from the Data Collection Tool (henceforth referred to as 'DCT notification') along with every regular notification that pops up (Fig. 2)

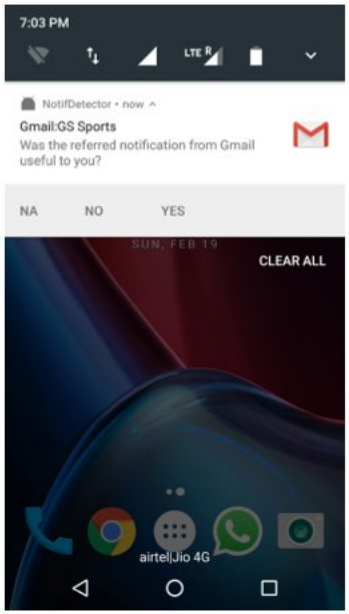


Fig. 2: Sample DCT Notification

For the user to be able to identify the notification that the DCT notification is referring to, it is important to give some sort of a connection between the two. Hence, an attempt was made to position the DCT notification directly below the referred notification so as to maintain a positional reference. But due to the clubbing of notifications in Android N, this strategy did not work. Therefore, explicit information referring to the App name and title of the referred notification was included in the DCT notification as reference for the user. The user is asked if the notification referred was useful to them.

The user is given 3 options in the notification: 'Yes', 'No' and 'NA'. 'NA' option is applicable only to the data collection phase, where system notifications like battery, wi-fi, alarm clock, etc. can be exempted from the notification tracking in order to reduce the load on the user.

The notifications that are marked 'NA' are added to a filter that ignores such messages in the future, i.e., does not post DCT notifications for them. The DCT, then uploads the collected data into an online firebase database in JSON format.

4.2.3 Collected Data

The collection tool was shared with 8 users (7 PG students, 1 Retired Professional). Data was collected over a course of 7 days. This data would act as reference for designing the machine learning algorithm.

At a glance, the data reveals some preliminary information:

The average number of notifications received per user per day was 71. Not all the notifications were responded to, by the users (since sometimes, it becomes too much effort to respond to all) Amongst the notifications with recorded responses, it was found that 47% were marked as irrelevant. This is an important indicator towards the number of low-priority notifications a user gets a day, which would be around 35.

Another interesting observation (which may not bear much in terms of the project) was that 50% of all the recorded notifications were from WhatsApp and out of them, 50% were from group messages.

4.2.4 Data Analysis

This step involved running a python code on the server (in this case, the developer's PC) to analyze and modify the data to the required format. The first steps were to prepare the data to a format that can be fed into the machine learning algorithm. While this involves mostly categorization for the non-textual data, it is more complex for the textual data.

Currently there are no reliable API's that run machine learning algorithms directly in a mobile device (mainly due to processing resource constraints). For the primary data collection step, all the data pertaining to each notification (as mentioned in 4.2.1) was uploaded to the server as is. But sending notification data directly over the internet to a third party's server raises security concerns. This is especially applicable to the title and content of the notification, since they can potentially contain sensitive information. Hence, the ideal way to manage the data would be to store the individual words from the content in the user's device itself; index them in an array and send the index values back to the third party server (preferably google machine learning API itself).

An example is given below:

Notification Title: "Ashok IIT"(In this case, the contact name)

Notification Content: "Meeting has been postponed to wednesday"

In this case, the title will be added to the title array as an item:

Title array before receiving the message : {"title1", "title2", "title3"}

Title array after receiving the message :
{"title1", "title2", "title3", "Ashok IIT"}

Now "Ashok IIT" will be referred to as 3 (index of the item in the array)

For the content, the following actions will be performed:

1. Remove stop words (commonly used words. Eg: “is”, “the”, “has”, etc.). This step basically helps to keep the data compact and relatively manageable for storage and analysis purposes.

Input: “Meeting has been postponed to wednesday”

Result: “Meeting postponed wednesday”

2. Stem the resulting words to their roots. This step, again helps make the data compact. Eg: “driven” or “drove” become “drive”

Input: “Meeting has been postponed to wednesday”

Result: “meeting postpone wednesday”

3. Split resulting words into single strings and add to content array (if it doesn’t already exist in the array.

Content array before: {“content1”, “postpone”}

Content array after:
{“content1”, “postpone”, “meeting”, “wednesday”}

Overall, the notification

Notification Title: “Ashok IIT”

Notification Content: “Meeting has been postponed to wednesday”

becomes

Notification Title: 3

Notification Content: {0,1,1,1}

One limitation of this system would be that if the content is in another language(eg: Hindi, Malayalam, etc.), then the above steps won’t work, since stemming is more complex in such cases. And skipping the stemming would result in a huge content array which would in turn require more processing power to sift through (which needs to happen every time a notification is received). It may be noted that there are languages (mainly ones that are not agglutinative), where all three steps would work. But for the purpose of this project, if the language is not english, then the content is just marked as “NA”.

4.3 Design Implications

In this section the different insights and design implications that came out of the primary and secondary research data, are discussed. The main inputs, here, are derived from the user statements and behaviour.

4.3.1 Types of users

The primary and secondary research pointed to mainly 4 types of users types:

1. User who doesn't customize at all : The user doesn't consider notifications as something that affects their work flow, though they do consider irrelevant notifications bothersome (especially promotional messages). There are two kinds of users under this category:

    a. User doesn't have a good conceptual model of the android system and hence tries to not fiddle around with the settings out of the fear of messing things up.

    b. User has a proper conceptual model of the android system and is aware of how to mute/block apps, but finds it too tedious to do it and hence does not.

2. Users who do moderate level customization: They know that fine adjustments can be done. They block apps that they find bothersome. But they don't customize at contact level, i.e., block notifications from individual contacts. This is due to two main reasons:

    a. Don't know that they can block at contact level

    b. Know that they can. But haven't tried because it's too deep within the settings

3. Users who do heavy customization to get exactly what they want (within the limitations of the system). These users tweak specific details to customize their system to their preferences. For example, they block specific contacts from messengers to avoid spam/irrelevant interruptions.

4. Users who put their phone on silent/priority mode, thereby blocking all incoming notifications during the period that they are engaged in some important activity. They still don't do a lot of customization. Their approach is more of cluster blocking/ muting as opposed to category 3.

4.3.2 Cater to needs of each type of user

In order to give power to each type of user, the system needs to have multiple levels of interactivity. At the same time, care should also be taken to not make the first level of interface too complicated. Else, users (type 1 and 2) may not use it at all.

4.3.3 'Filter. Not Block.'

An interesting observation was that these user of type 1 and 2 still muted selective WhatsApp chats. While messenger also works in a similar way, the user doesn't do the same there. A key difference between the two cases is that WhatsApp chats(muted ones) will still be seen by the user when they check their feed for other messages or when they feel like checking them. But the blocked notifications are not looked at by the user(since most users rarely checked their 'Messenger' feed. Coupling this with the observation that 'Users were afraid of missing out on useful/important messages', the following design insight was drawn: 'Filtering notifications is better than blocking them altogether. The option for blocking should also be there, though'. The filtered notifications need to be shown to the users periodically so that they don't miss out on anything(relevant or not). This could potentially encourage users to actively mute more notifications, which would directly fuel the aim of this project.

4.3.4 Prompt Feedback to Users' Actions

In order for the users to keep using a feature, they need to see that their actions are making a difference. In this case, if the user marks a particular notification as irrelevant, they expect the next notification, that is the same as the one they marked, should get filtered out (Finding from the user interviews). Hence, the system should be designed in such a way that there is prompt feedback to the user's action.

4.3.5 Delay Filtered Notifications

Secondary research revealed an interesting user behaviour where the user would check their phone for notifications even if they set their phones on silent mode. This is a possible side effect of muting the messages. If the user checks their phone at some interval and keeps finding filtered messages, it could potentially develop a habit of checking. This would end up breaking the flow of work of the user, which is quite the opposite of the aim of the project. The design implication of this point was that the notifications that get filtered need to be deferred and shown to the user as a bunch rather than as and when they pop.

Based on the findings from the primary research, the planned final output was:

- Interactions: For user to communicate preferences to the system
- Presentation of notifications
- A framework for predicting relevance of future notifications based on data collected from past notifications (Back end)

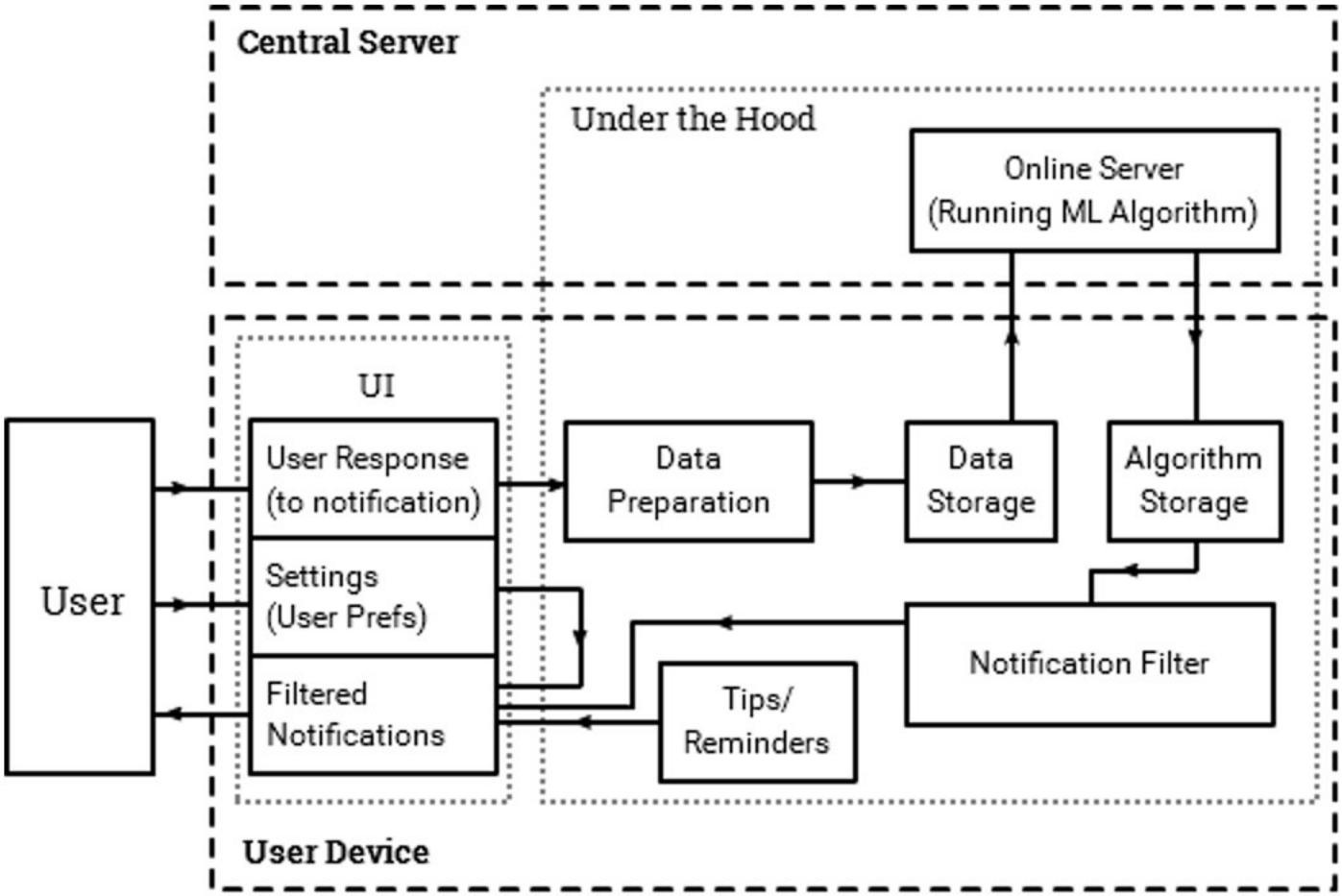


Fig 3. System Level Diagram

5. System Design and Ideation

There were two main points of direct interaction between the user and the notification system:

1. User responses indicating the relevance of a notification
2. The presentation of the filtered notifications to the user

The rest of the working of the system is under the hood. The approach to this stage was to explore different features and interactions that contribute to the goal of the project (to reduce the number of interruptions) by catering to points that emerged from the primary and secondary research stages.

Based on the prior steps, a system was developed (as visualized in Fig 3) The following sections discuss in detail the function and working of the different modules of the system.

5.1 User Interactions - User Response

The proposed notification system requires input from the user in order to train itself to understand which notifications interest and conversely, which don't. In other words, explicit response from the user as to the relevance of a notification is a requirement for the system to work. One direct response indicating a notification as relevant would be if the user clicks it. But the ones that the user doesn't click on may not be irrelevant. (For example, One Time Password (OTP) messages, where the message is such that the user can get the OTP without having to open the message). In the primary research stage, this information was collected through an additional notification (for each notification) using which the user could rate the original notification's relevance (as 'useful' or 'not useful'). But such a method is not viable as a permanent solution, since it is basically adding to the base problem (too many notifications). Therefore, there is a need to find better ways for the user to interact with the notification system.

Another point of interest was the presentation of the filtered notifications to the user. While the filtered ones are of low relevance to the user, the aim is to give the user the power to ultimately dispose of them, rather than doing it for them. The logic behind this approach was to give the user, at all times, the power to manually override the relevance assigned by the algorithm. This is especially important in the event of a failure on the part of the filtering algorithm causing a relevant notification being filtered out. Hence, there is a need for the filtered notifications to be presented to the user.

In this stage another round of secondary research was done to explore the different kinds of interactions available to the smartphone user (relevant to notifications). This was followed by ideation for exploring different interactions that can be used to collect the required information; and the different ways to present the filtered notifications to user.

The following are the different interactions that were considered:

5.1.1 Click + Swipe left/right

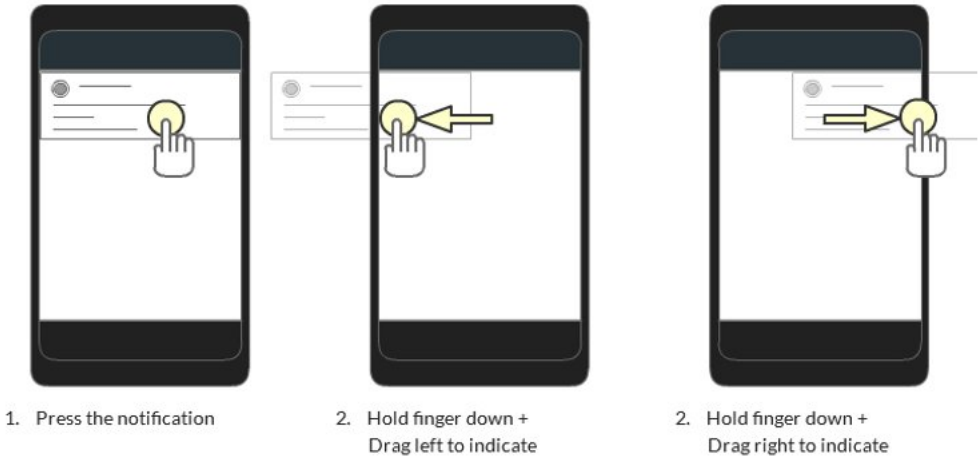


Fig 4. Click + Swipe left/right

Number of actions: 1(Click) + 0.5 (Swipe) = 1.5

Pros: Practically effortless

Cons:

- The user gets forced to respond to pretty much all the notifications.
- In some cases the user may not be sure of whether they want to swipe left/right (since either action would have a consequence). This could put unnecessary cognitive load on the user

5.1.2 Expansion of Android N's Notification Setting

A checkbox for marking the notification as irrelevant provided as an extension of the Android N notification settings.

Fig 5. Interaction idea 2

Number of actions:

1 (Click)+ 0.5 (Swipe) + 1 (Click) + 1 (Click) + 1 (Click) = 4.5

Pros:

- The text enables explicitly let the user know what the checkbox does.

Cons:

The below points may dissuade users of Type 1 and 2 from using it

- Too many actions required.
- Too much data/options presented to user on the same screen.

5.1.3 Expansion of android N's notification setting

Similar to 5.1.2, but the action required to bring up the settings is to click and hold the notification (again, similar to android N's interaction)

Number of actions: 1 (Click)+ 0.5 (Hold) + 1 (Click) + 1 (Click) = 3.5

Pros:

The text enables explicitly let the user know what the checkbox does.

Cons:

- The below points may dissuade users of Type 1 and 2 from using it
- Moderately high number of actions required.
- Too much data/options presented to user on the same screen.

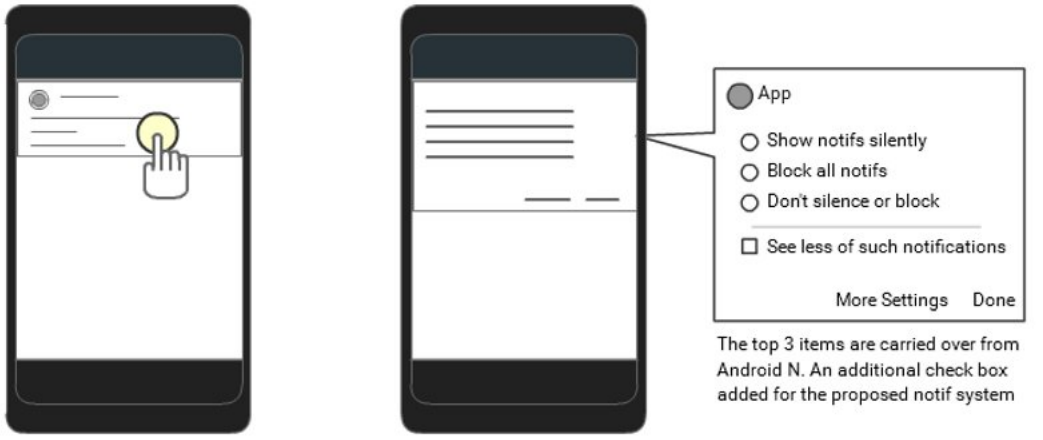


Fig 6. Interaction idea 3

5.1.4 As an extension of the notification

Fig 7. Interaction idea 4

Number of actions required:

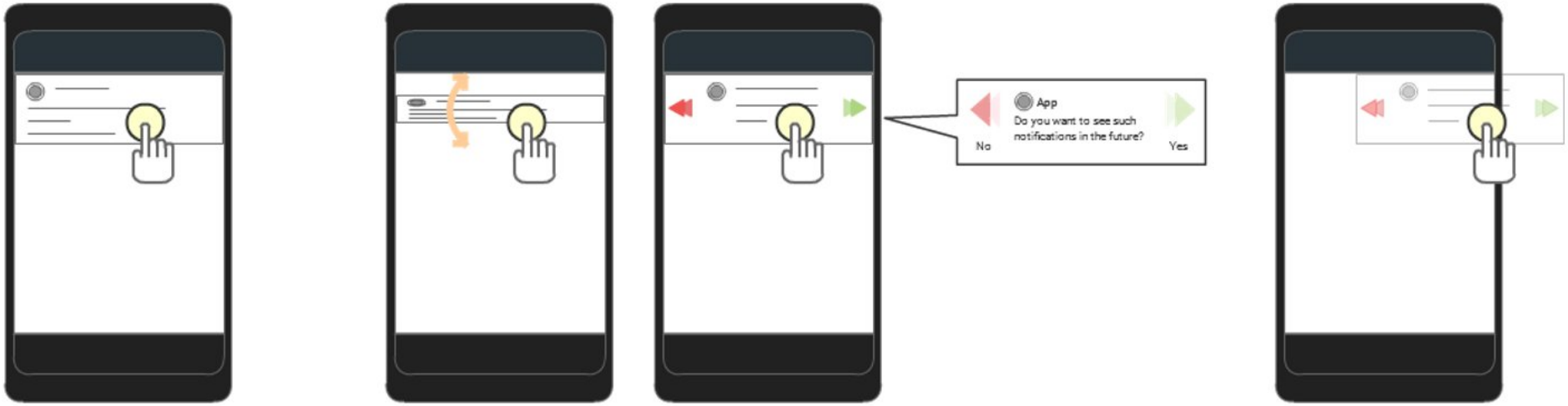
1 (Click) + 0.5 (Swipe) + 1 (Click) + 0.5 (Swipe) + 1 (Click) + 0.5 (Swipe) = 4.5

Pros:

Low cognitive load (Only two options (yes/no) presented to user)

Cons: Too many actions required.

5.1.5 Flip and swipe



1. Press and hold notification
2. Hold notification till it flips (Vibration feedback)
3. Hold finger down + Swipe right or left to indicate high/low relevance

Fig 8. Interaction idea 5

Pros:

- Goes with the card metaphor that android pushes. (The information is given on the flipside of the card)
- User has the option of not responding at all (by just swiping the notification away)
- Takes only 2 actions to respond. Hence, more likely to be used by Type 1 & 2 users
- The text enables explicitly let the user know what their action does

Number of actions : 1 (Click) + 0.5 (Hold) + 0.5 (Swipe) = 2

This interaction was selected for the further steps.

5.2 Presentation of Filtered Notifications

Based on primary research, a user could get >40 filtered notifications a day. Since most of these notifications would be filtered out and presented to the user in one go, the layout has to be modified to suit the same. Currently, android system groups notifications based on the source app. Here, a grouping based on the type of notification is proposed:

The primary groups would be “Messages”, “Promotions” & “Updates” (Much like the gmail grouping).

The same grouping is not carried over to unfiltered notifications, since the underlying assumption is that the unfiltered ones are important and should be presented to the user without the need for any extra steps to read them.

Next, the flow for the presentation of the grouped notifications was designed and wireframe was developed. (Fig. 9)

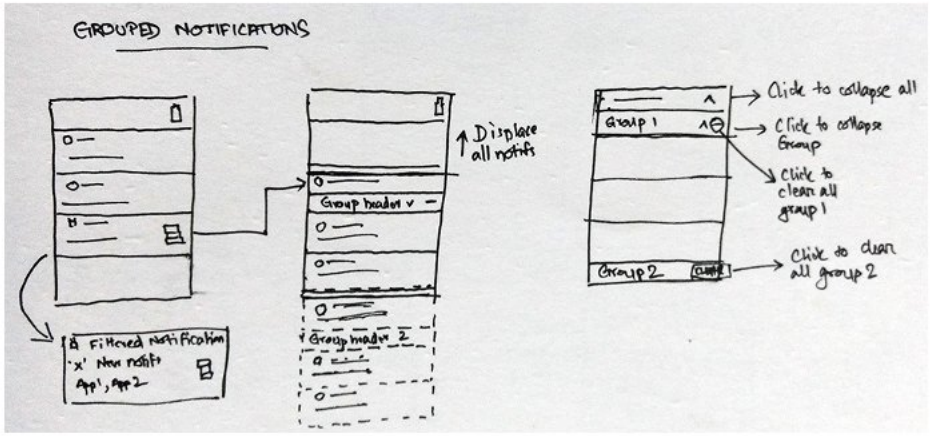


Fig 9. Grouped Notification Wireframe & Flow

5.3 Finer Controls : Manual Grouping

Derived from insight: 4.3.1 & 4.3.2

Type 3 users are likely to desire more control over the grouping than just stick with the default ones discussed in section 5.2. Hence, provisions for creating custom groups and adding apps to the group list would be made. Subsequently, the flow for the grouping settings was designed and wireframe was developed (Fig. 10).

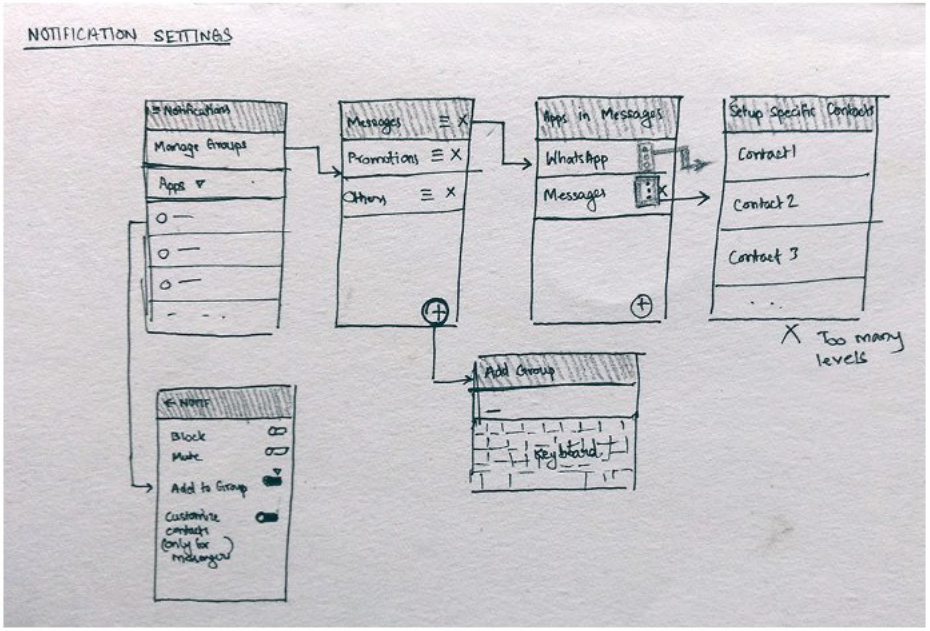


Fig 10. Fine Control Ideation Wireframes

5.4 Feedback to Users

Derived from insight: 4.3.4

For the system to be continuously used by the user, there needs to be direct and visible effects to the user's actions (in this case, the action refers to marking of notifications as relevant or not). Without such feedback, the user is likely to stop using it eventually. This point is based on the primary research, when the users(on whose phones the Data Collection Tool was installed) were asked about what they expect the system to do. One common response (from users of any category) was that they "expect the notification that they mark irrelevant to get filtered from the next time onwards".

The long term feedback on their actions could be through periodic notifications from the android system informing the user about how many interruptions were filtered out by the system during that time period. This could potentially motivate the user to keep giving feedbacks to the future notifications.

5.5 Reminders/ Tips

User could be given notifications (filtered) teaching/reminding them how to use this feature. These notifications would go under filtered notifications.

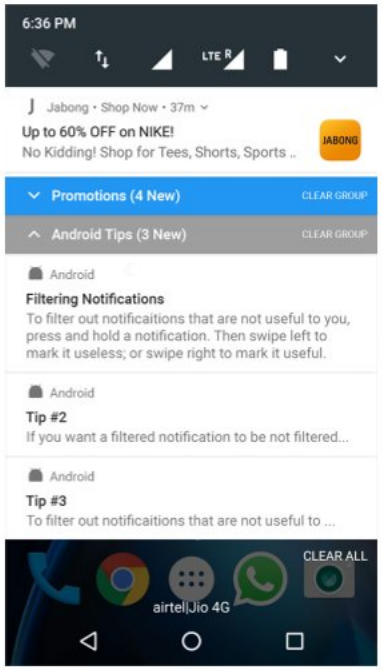


Fig 11. Tips

6. Prototyping

Since the final product that is being pitched in this project is an expansion on the Android OS, making a fully functional prototype is not an option . The reason why the entire system cannot be prototyped into one functional prototype was that it would require modifying the android OS's core codes to introduce the new interactions to the notifications. Hence the whole prototype was split into UI and system. The UI part is meant to demonstrate the working of the new interactions and the system is meant to demonstrate the working of the Notification FIltering algorithm. The system prototype would also be used to check how effectively and accurately the algorithm filters out the notifications.

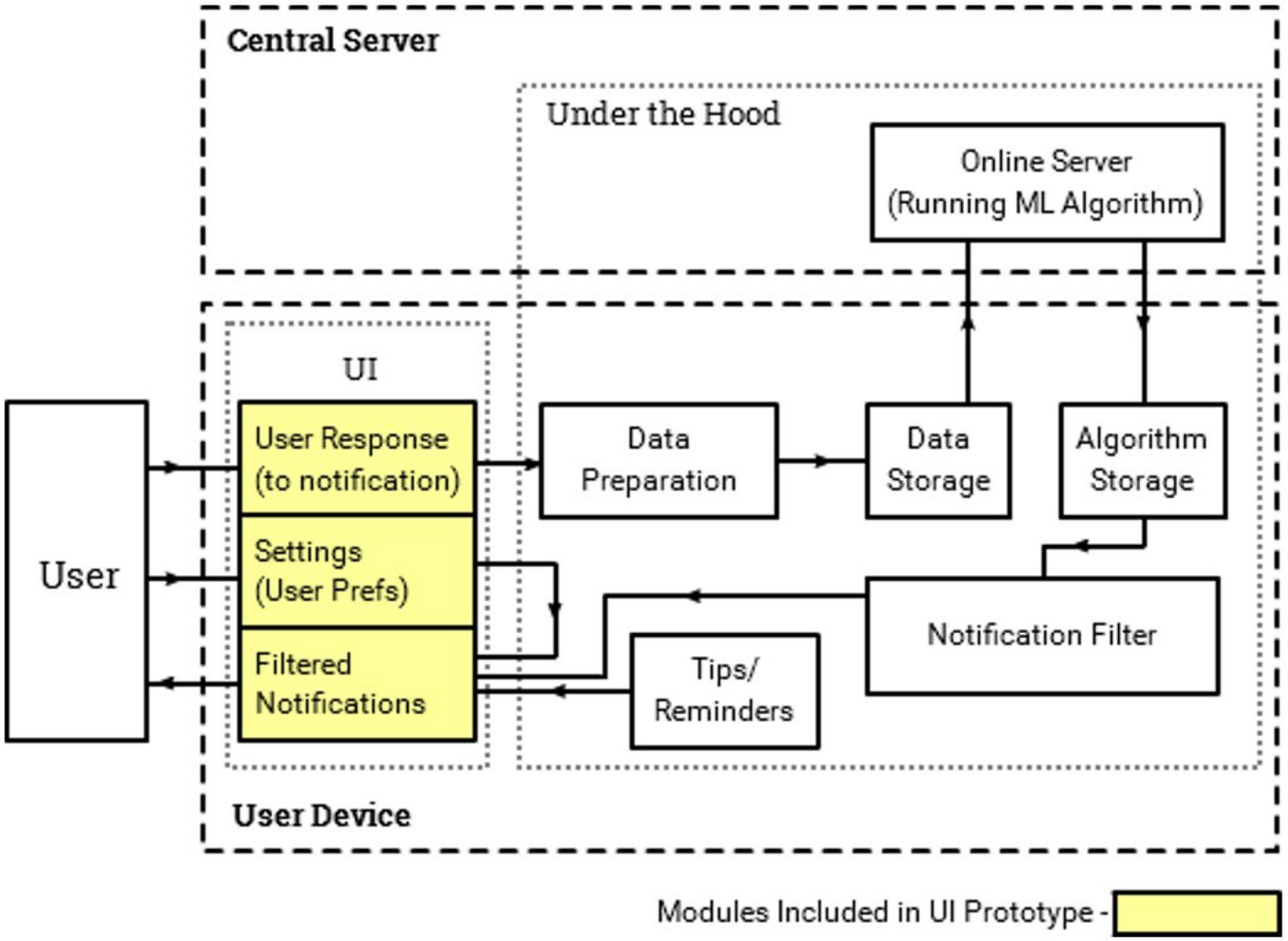


Fig 12. UI Prototype Overview

6.1 UI

The UI prototype includes the custom interactions, settings and presentation of the filtered notifications (Fig. 12). The purpose of this part of the prototype is to be used for heuristic evaluation of the different UI components. It was developed using Adobe Flash and actionscript 3.0. The application mimics the notification bar of Android, i.e., the user can interact with the sample notifications provided in the prototype similar to the normal android system but along with the new interactions that were introduced for this project.

UI Screens

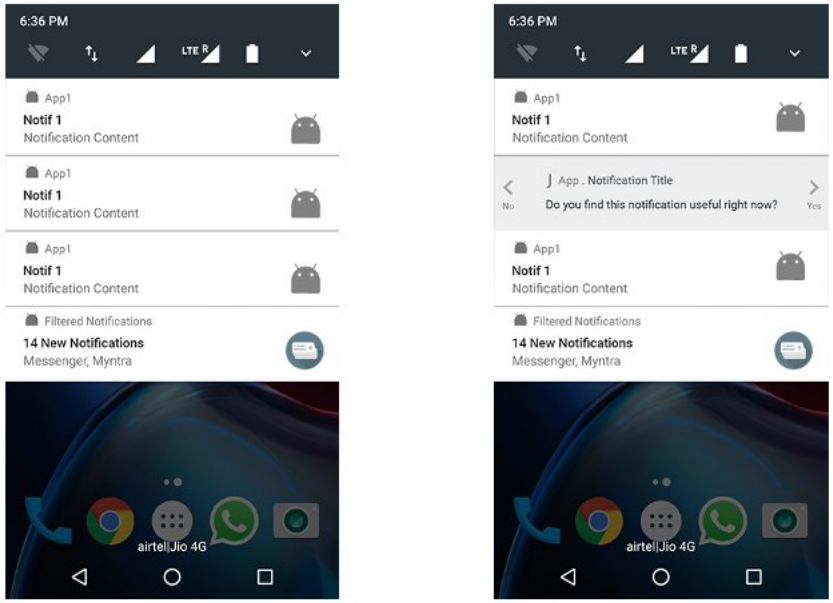


Fig. 13 Individual notification interactions

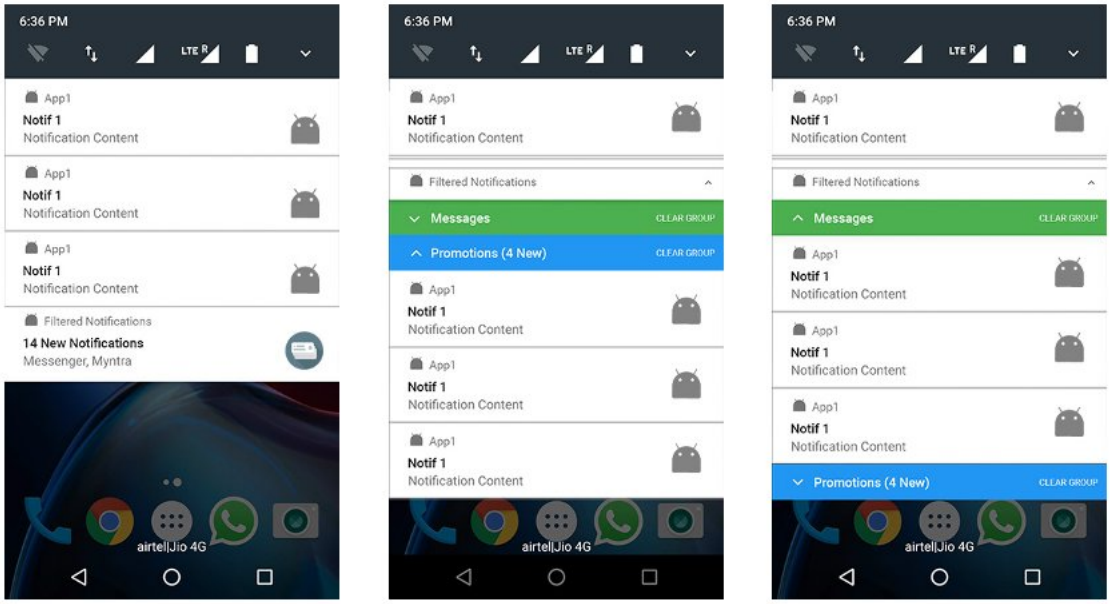


Fig. 14 Group notification Interactions

Fig 15. Icon For Grouped Notifications

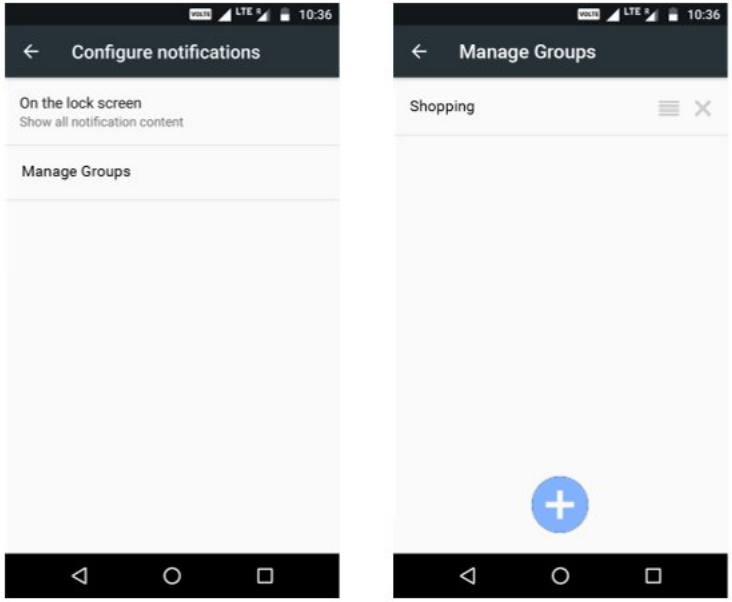


Fig 16. Fine Control

6.2 System Prototype

The system level prototype includes an Android application that collects the notification information; and an online server that applies machine learning on this data to classify notifications as relevant or not (Fig. 17). This part of the prototype will be used to evaluate the performance of the overall system in terms of its prime purpose, which is to reduce interruptions for the user. Here, the performance pertains to the system's ability to identify irrelevant messages. The development details of the system prototype has already been discussed in more detail in section 4.2. The final system design involved modifications and updations on the data collection app to get the desired outcome; the desired outcome in this case being 100% correct classification (as relevant or irrelevant) of all notifications. While that may not be possible with limited amount of data (which could be the case if the user doesn't actively mark the irrelevant notifications), the algorithm had to be tweaked in a way that it would never classify an important notification as irrelevant. This is to ensure that the user does not miss out on important notifications on the system's account. This is of utmost importance because if the user finds important notifications inside the filtered notifications (even once), they may start checking the filtered ones all the time to ensure that they're not missing anything, which would beat the purpose of the whole system to begin with.

6.2.1 Filtering Algorithm

The filtering algorithm is what takes the user's input (that come through the interactions discussed in section 5.1), learns from this input and predicts if the next notification would be relevant to the user or not. For this purpose, machine learning(ML) was employed. The algorithm training is done on the online server. An ML algorithm needs to be trained before it can identify a relevant/irrelevant notifications. The data uploaded from the user devices are used as the input used to train the ML algorithm.

The filtering algorithm, overall consisted of three main stages:

1. Data preparation - The stage where data is arranged into a format that can be used to train the machine learning part
2. Machine Learning Algorithm training - The formatted data is used to train the ML model. After that, the ML model is ready to filter notifications.
3. ML Filtering - In this stage, the trained ML model is used to predict the relevance of future notifications.

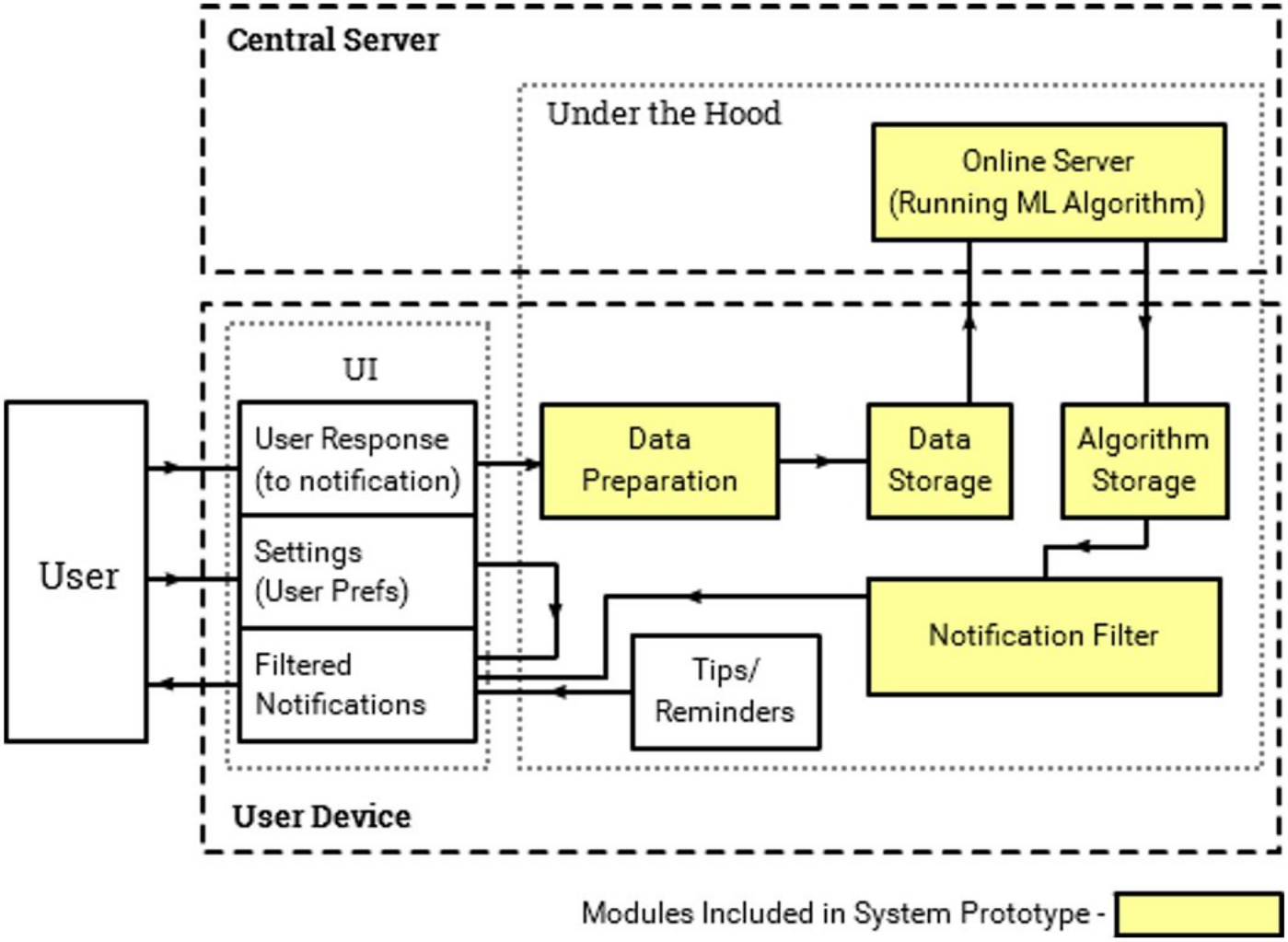


Fig 17. System Prototype Overview

Sklearn library (for python) was used for the ML part of the algorithm. Different ML algorithms were tried out and the one which showed the best base accuracy was chosen. Based on the requirement of the application, the ML technique that needs to be used here is classification[15] using supervised learning[15]. The different ML techniques tried out were: Naive Bayesian(NB) Classification, Decision Tree and Support Vector Machines (SVM). The algorithm was trained using 60% of the data collected during primary research. It was then tested against the remaining 40% to check for accuracy. The numbers for the different ML models were as below:

With 234 training data points:

ML Model	Accuracy (%)
Naive Bayesian Classifier	62
SVM	88
Decision Tree	83

Table 1: ML Model Accuracy with 234 Training Points

ML Model	Accuracy (%)
Naive Bayesian Classifier	59
SVM	87
Decision Tree	91

Table 2: ML Model Accuracy with 1163 Training Points

On training with more data, some models started showing better accuracy (Table 1 & 2).

Since it had the best overall accuracy, the decision tree model was chosen for further steps.

Overrides: In accordance with the design insight (Section 4.3.4), the user needs immediate feedback to their actions. But ML doesn't make a direct judgement based on a single data point. Hence an override code was added which would check if a notification is the exact same as one before (that has a user response). If the notification was marked as irrelevant by the user, then all future instances of notifications with the 'exact same' features as that notification would be marked irrelevant and filtered out.

In the above steps the filtering algorithm is completed. But the same needs to be exported back into the user device. The other option is to upload the notification data to the server every time the user device receives a notification; evaluate the predicted relevance of the notification on the server; and send it back to the user device. But this would require data connectivity for the system to function. Instead, the filtering algorithm was exported back to the user device by running all permutations of variables (on the server) through the filter and sending back the ones that returned irrelevant. This dataset was then stored in the user device. For every notification that gets received, the system would check it against the returned dataset and filter based on the same.

7. Evaluation

The system level performance of the system is indicated by the accuracy with which the system can identify irrelevant notifications. This was checked by collecting the notifications that are received by the user's device and running them through the algorithm that was trained with the data that they input during the primary research phase. The metric for evaluation are the accuracy of classification (the percentage of filtered notifications that were actually irrelevant) and the rate of filtering (The ratio of identified irrelevant notifications to total irrelevant notifications). While the rate of filtering would keep improving as time goes on (and the system learns the user's preferences), the accuracy of classification should ideally be close to 100%.

The evaluation of the overall system was split into two major parts:

1. Long term performance evaluation:
2. Short term performance evaluation:

7.1 Long term performance evaluation

This stage involved the simultaneous design and evaluation of the ML algorithm. The method followed for the evaluation was train/test split. This involved using the data collected using the DCT (section XX) being used for both training and testing the algorithm. Out of all the users that the data was collected from, the one that was collected over 1 month was used for this purpose, since it had the most number of data points. The dataset was split into two, one with the first 60% of data points (training set); and other with the remaining 40% (testing set). All of these data points had the user response w.r.t relevance. The training set and testing set were used for training and testing the algorithm respectively. The different ML methods mentioned in section Section (6.2.1) were tried out. The performance details are presented in Fig 18.

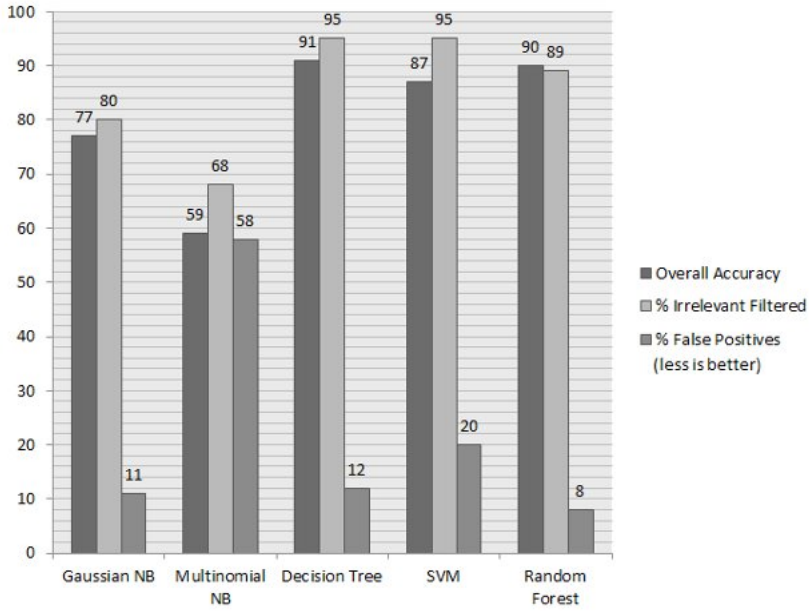


Fig 18 : ML Algorithm Performance

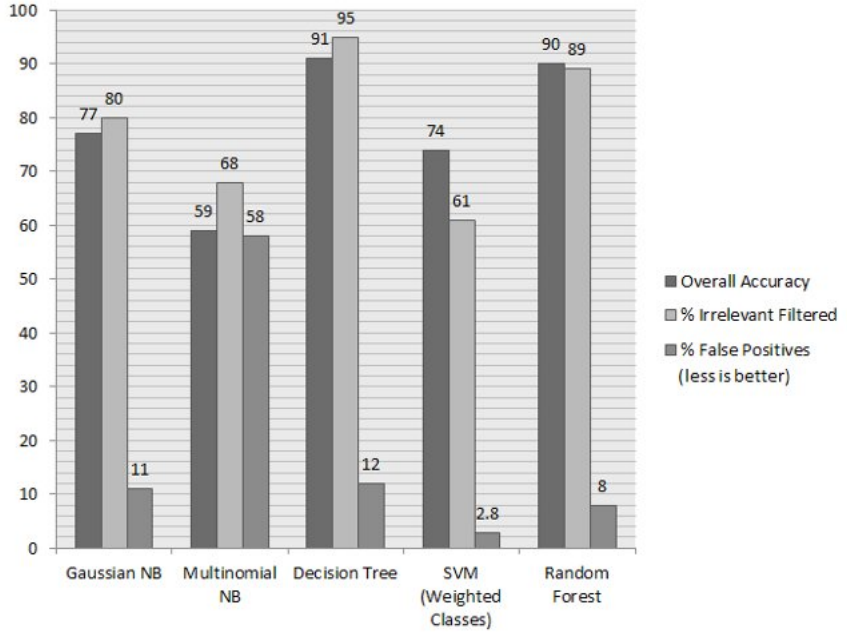


Fig 19: Second Round Evaluation Performances

Decision Tree, SVM and Random Forest performed very well on the overall accuracy and '% irrelevant filtered'. But the most important metric, which is the '% false positives' was still too high. It should ideally be 0. Hence tweaks were attempted on the algorithms. Weighted classes were used on SVM and the test re-run. The performance was as given in Fig 19. While SVM's filtering rate and overall accuracy came down, it's '% false positives' went all the way down to 2.8%. Hence SVM and Random Forest were chosen as possible candidates for the ML algorithm.

In order for the algorithm to perform in low-data conditions (at the initial stages of use), an add-on code was written, which looks for exact matches of notifications that have already been marked by the user and replicates the response for the next one of the same kind. This helps give give immediate feedback to the user so that they keep using the feature (Refer Section 4.3.4). This was tested out in the section 7.2.

7.2 Short term performance evaluation

This part of the evaluation was more of an indicative one. The main purpose being to check how the system performs when there isn't sufficient data to make accurate predictions. This was done using a prototype installed in the user devices. The prototype collected the user preferences using a separate notification, similar to the DCT. This data was uploaded to an online db and downloaded fed into a python framework which updated the filtering criteria for the user's phone. This system was evaluated with 4 users over the course of 1 day. The system started filtering in the first 4 hours of the evaluation period. There were no instances of filtering out relevant messages. But the overall filtering rate was below 40%. On a subjective front, the users found the filtering to be useful. But the fact that the filtering mechanism was a black box, (where they couldn't have sufficient control over the filtering criteria) made the users cautious about responding to the app. This would not be an issue with the features mentioned earlier.

8. Future Scope

The prototype developed for this project demonstrates the UI and the underlying algorithm of the system separately. The next step would be to implement the complete system (UI and algorithm) atop android OS and test it out to see how it performs. Since the system relies on the user feeding it data in order to learn about their preferences, the user needs to be using the interactions afforded by it. While there are elements added in the design (namely 'an easy to use interaction to respond' (section 5.1), 'feedback' (section 5.4) and 'tips/reminders' (section 5.5)) to help ensure that the user utilizes this feature, it needs to be checked whether they are enough to nudge the user into actively use the system.

The system, in its current form, does not use information from sensors like accelerometers, gyro, GPS, etc. due to the fact that they end up consuming too much of the phone battery. If this issue can be resolved (at OS or hardware level), these factors can be used to model a much more robust system. Similarly, using the calendar information of the user can be another way to make the system more robust.

9. Conclusion

The project started out with the aim of reducing the number of irrelevant interruptions in the lives of the smartphone users. During the course of the project, a smart notification system was designed, prototyped and evaluated against the aim of the project. The system included new interactions and features (to be added to the smartphone OS) that would enable the user to communicate their preferences to the system; and an algorithm that uses machine learning to predict and screen the irrelevant notifications. Two ML methods were employed for developing the algorithm: Support Vector Machine(SVM) and Random Forest. Evaluation of the prototype showed that the system reduced the number of interruptions by 61%(using SVM) and 89%(using Random Forest) after training with 18 days worth of data; with a false positives rate of 2.8%(using SVM)and 7.7 %(using Random Forest). While the overall framework has been defined, there is room for improvement in the algorithm, which can be further tweaked to improve the performance.

References

[1] Julie Aranda, Noor Ali-Hasan, and Safia Baig. 2016. I'm just trying to survive: an ethnographic look at mobile notifications and attention management. In Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct (MobileHCI '16). ACM, New York, NY, USA, 564-574. DOI: https://doi.org/10.1145/2957265.2957274

[2] Martin Pielot, Karen Church, and Rodrigo de Oliveira. 2014. An in-situ study of mobile phone notifications. In Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services (MobileHCI '14). ACM, New York, NY, USA, 233-242. DOI: http://dx.doi.org/10.1145/2628363.2628364

[3] Joel E. Fischer, Chris Greenhalgh, and Steve Benford. 2011. Investigating episodes of mobile phone activity as indicators of opportune moments to deliver notifications. In Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI '11). ACM, New York, NY, USA, 181-190. DOI=http://dx.doi.org/10.1145/2037373.2037402

[4] Jeremiah Smith, Anna Lavygina, Jiefei Ma, Alessandra Russo, and Naranker Dulay. 2014. Learning to recognise disruptive smartphone notifications. In Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services (MobileHCI '14). ACM, New York, NY, USA, 121-124.
DOI: http://dx.doi.org/10.1145/2628363.2628404

[5] Hugo Lopez-Tovar, Andreas Charalambous, and John Dowell. 2015. Managing Smartphone Interruptions through Adaptive Modes and Modulation of Notifications. In Proceedings of the 20th International Conference on Intelligent User Interfaces (IUI '15). ACM, New York, NY, USA, 296-299. DOI=10.1145/2678025.2701390 http://doi.acm.org/10.1145/2678025.2701390

[6] Veljko Pejovic and Mirco Musolesi. 2014. InterruptMe: designing intelligent prompting mechanisms for pervasive applications. In Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '14). ACM, New York, NY, USA, 897-908. DOI=http://dx.doi.org/10.1145/2632048.2632062

[7] Jeremiah Smith and Naranker Dulay. 2014. Ringlearn: Long-term mitigation of disruptive smartphone interruptions. In Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014. IEEE International Conference on, IEEE (2014), 27–35.

[8] Fisher and Simmons. 2011. Smartphone interruptibility using density-weighted uncertainty sampling with reinforcement learning. In Machine Learning and Applications and Workshops (ICMLA), 2011. 10th International Conference on, vol. 1, IEEE (2011), 436–441.

[9] SungHyuk Yoon, Sang-su Lee, Jae-myung Lee, and KunPyo Lee. 2014. Understanding notification stress of smartphone messenger app. In CHI '14 Extended Abstracts on Human Factors in Computing Systems (CHI EA '14). ACM, New York, NY, USA, 1735-1740. DOI: https://doi.org/10.1145/2559206.2581167

[10] SungHyuk Yoon, Sang-su Lee, Jae-myung Lee, and KunPyo Lee. 2015. A Study On Notification System Design Of Smartphone Messenger Considering The User's Stress. Archives of Design Research 28.2 (2015): 75. Web.

[11] Ray, S., Shaikh, F., Srivastava, T., & Kashyap, S. (2017, January 31). Essentials of Machine Learning Algorithms (with Python and R Codes). Retrieved Mar 12, 2017, from https://www.analyticsvidhya.com/blog/2015/08/common-machine-learning-algorithms

[12] Supervised learning¶. (n.d.). Retrieved March 12, 2017, from http://scikit-learn.org/stable/supervised_learning.html#supervised-learning

[13] Supervised learning¶. (n.d.). Retrieved March 12, 2017, from http://scikit-learn.org/stable/supervised_learning.html#supervised-learning

[14] Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: A review of classification techniques.