USING MACHINE LEARNING AND BEHAVIOURAL ANALYSIS FOR USER-TAILORED VIEWER EXPERIENCE

Peter W. Szabo and Zsolt L. Janosi

3 Screen Solutions (3SS), Romania

ABSTRACT

Currently, user interfaces displayed to viewers on their TVs look and behave similarly for all users. While sometimes it is possible to customise the user interface (UI) to a degree, users will rarely experience true customisation on a TV, mainly because that is difficult using a remote control or voice commands. Our research focuses on utilising machine learning to discover and interpret behavioural patterns and to adapt the UI accordingly. In this paper, we will share our solution for a truly adaptive UI, tailored to each viewer. This paper also showcases the machine learning engine, and examines our behavioural mapping technique and the mathematical theory behind it.

INTRODUCTION

After years of researching digital television user experience (UX), we must accept that there is no simple and easy way for the viewer to customise the user interface. Even if there were a way for the consumer to achieve this, we have found a better, more powerful solution. What if the software could understand and learn what a specific viewer wants and adapt accordingly, as shown in Figure 1?

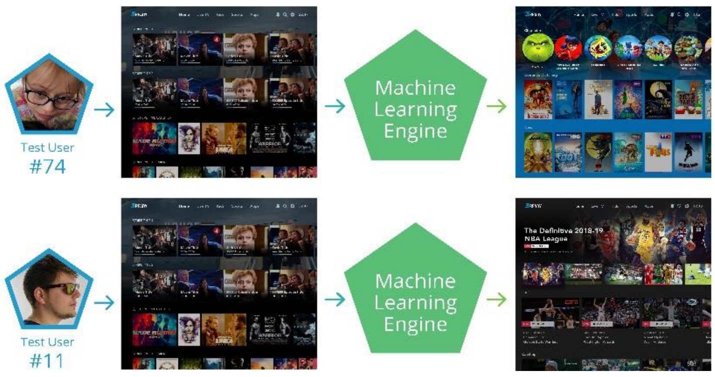

Figure 1 – AI customised UX

Personalisation of the digital television user interface has been identified as a crucial research topic, but most researchers to date have focused on pre-customised UI design and standardisation. 'Joonhwan et al (1)', 'Young (2)'. The problem is that a standardised user interface will never provide an ideal user experience for each individual in the entire viewing/subscribing audience base because the needs and behaviours of each user can be vastly different from one another. A preschool child will have a very different mental model and thought process from an elderly couple watching TV together, or a young adult for example. Recent research in user experience increasingly emphasises viewers' mental models, so the focus is shifted away from designing a solution towards understanding the user's state of mind, and how we, or the service provider, can support those states. 'Szabo (3)'

It is obvious that the same user experience will not satisfy all users. This is why we need a way to ease customisation. 'Böhmer and Krüger (4)'. If this is the case, how is it possible that this problem has not yet been solved? The remote control itself is not ideal for the task. There have been suggestions for using gestures, pressure and breath as interaction mechanisms 'Bernhaupt et al (5)' for interacting with a TV. Voice control seems to be a better approach, but it is still not ideal, easy to use or powerful enough. 'Papp et al (6)'. In recent years, user experience research has focused on using machine learning to create audience segments or personas. This still implies designing an experience for each persona manually, but it is a step in the direction of UI customisation. 'Triolo et al (7)'

THE TELEVISION EXPERIENCE SUCCESS

For a behaviour to occur, we need a trigger. If the trigger is present, we need a certain amount of motivation and ability. It has also been demonstrated that if the motivation is high, the behaviour can occur even at low ability values, and vice versa. 'Fogg (8)' for example, if it's hard for the user to find a specific movie, s/he is likely to pick something similar which is easier to find, unless they strongly desire that specific movie. Figure 2 shows the behaviour graph, with the activation threshold, above which the behaviour occurs.

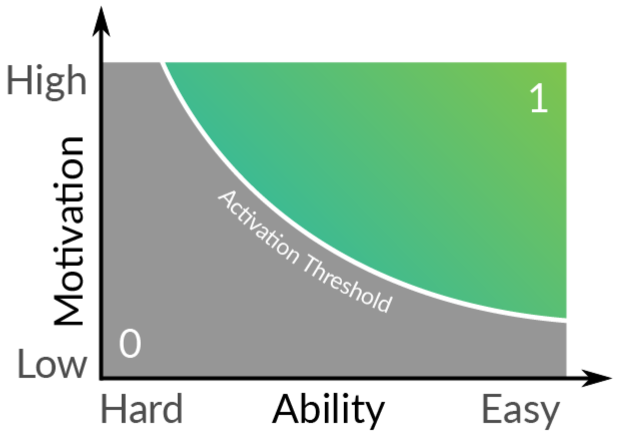

Figure 2 – The activation threshold

Although we can't measure the user's motivation directly, we know the information architecture. We can assign a numeric value to 'ability' based on the user journey (number of steps and time elapsed). This information, when used in the behaviour model, can be used to calculate the estimated 'motivation'.

Ultimately, we are interested in the mental model of the individual user. We need this understanding in order to create a user interface where we maximise the user's ability to execute their tasks. For example, if our model suggests that a viewer often spends several minutes trying to find movies from the '80s and/or action movies, we can reduce this time by creating a stripe on the home page of the digital TV application which highlights films in these categories by displaying them in top/most prominent positions. We might also set the home page background to be an iconic scene from such a movie in order to produce the trigger. If this stripe contains movies that can be purchased or rented, then we increase the conversion rate of our solution, while also increasing the customer's satisfaction.

APPROACH

In order to create the best possible user experience for user A we record all behaviours. This means that we start recording the behaviours of A_β as an array of behaviours. $A_\beta = [\beta_1, \beta_2, \dots, \beta_m]$ behaviours. We define a single behaviour (β_x) as a one-dimensional array of many individual ε_x events. So $\beta_x = [\varepsilon_1, \varepsilon_2 \dots, \varepsilon_n]$.

This approach led to the topic of the cut-off. In other words, when does one behaviour end, and when does the next behaviour start? We could have defined the behavioural tensor as a single behaviour, constantly adding new events, essentially making $A_\beta = \beta_1$. That approach would have been quite detrimental to improving user experience, therefore we had to introduce the cut-off.

Definition 1: The cut-off is an event at which the current behaviour ends and after which a new one starts. We defined that cut-off happens at the time when the success of the user journey is measurable.

Success in the user's journey is when the problem the user initially had is solved. This drive for user experience success is essentially why we developed this solution. 'Lichaw (9)'

For digital television the problem is the viewer's need to watch something relevant. We solve this by creating a user interface which enables finding and watching video content. Much of the time the users will have no clear idea of what they want to watch. This is why traditional recommender systems were created 'Adomavicius et al. (10)', 'Basu et al (11)'

One of the breakthroughs in recommendation engine research in 2019 was presented by 'Vijayakumar et al (12)' which employed a heat map of already-visited travel locations to create new travel recommendations for consumers, including multiple points of interest. That research focused on a domain unrelated to digital TV, but the most important learning for us was that departing from a rating-based recommendation system can result in significantly improved user experience.

We have defined cut-off as being when the user chooses to stop watching the selected item (movie, series, documentary, etc.). This means that we often need to re-evaluate the behaviour if the user resumes watching. Our goal is to create a user experience where the user can find an item to watch. So once the item of content is found, if the viewer watches it until the end, it means we were successful to a degree. But they don't necessarily have to watch it until the very last frame. Before the automatic cut-off, the user's actions can dispatch the cut-off event. While, for example, pausing the playback will not dispatch such an event, navigating away or searching for another item does.

Definition 2: Any user event with the intent of finding a different item creates a new behaviour, and the event chronologically before that will serve as the cut-off for this new behaviour.

Definition 3: The success of the television experience is the numerical representation of how successful our solution is in fulfilling the user's need to find something they want to watch while minimising both the amount of time spent before the start of the playback and the number of steps needed to reach playback.

As a result, we have created the television experience success equation, our approach to converting a complex user behaviour into a single numeric value. $\Omega_{\beta x}$ is the television experience success of the β_x behaviour.

$$\Omega_{\beta_x} = \frac{1}{t_s^2} C_\beta \, k \prod_{i=1}^{n} \Delta_{\varepsilon_i}$$

We want to emphasise the fact that optimising the time needed to find the desired item using the interface is crucial for a good user experience. We note the time from the behaviour's beginning until the start of the playback as t_s. We took the inverse of the square of this value because higher time taken substantially lowers user experiences. This way we avoid the pitfall of generating a very long list of recommended items in order to minimise the number of steps needed.

Let $C_\beta = 1$ if the automatic cut-off event triggered after $1-p$ ratio of completion of the item. Otherwise if there is a user-triggered cut-off, we set $C_\beta = \frac{1}{t_s}$. In other words, if the user finds the wrong item, at least they should not waste much time.

So if the user watches the item to a completion ratio less than $1 - p$, the equation gets simplified into:

$$\Omega_{\beta_x} = \frac{k \prod \Delta_{\varepsilon_i}}{t_s^3}$$

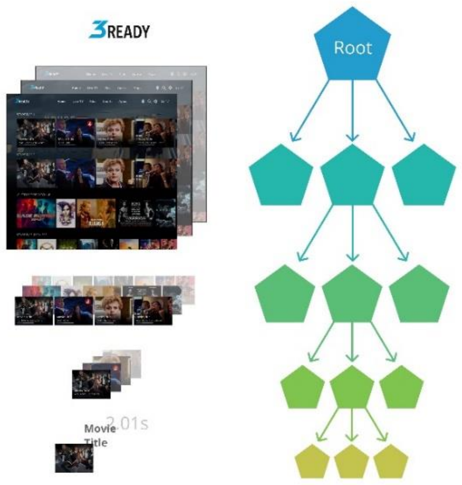

Figure 3 – The UI graph

Let Δ_{ε_i} represent the ability of the user to complete the ε_i event. By definition we restrict the ability to be in the interval of [0,1], where 0 means failure to complete and abandonment of the behaviour. As ability is the least important among the three variables, we introduced the k dampening factor, which can be used to dynamically fine-tune the importance of the product of event abilities in the equation.

ADAPTING THE UI TO INDIVIDUAL VIEWERS

Because our engine is basically a deep network on graph-structured data 'Henaff et al (13)', we had to find a way to describe the user interface as a graph, ideally a directed graph in which any two vertices are connected by exactly one path. Figure 3 is the visual representation of a tree graph. We applied the principles of atomic design 'Frost (14)' and mapped these elements into a JSON 'Bray (15)' structure.

The front-end applications need to build their implementation, and their rendering structure, based on the configuration that is loaded when bootstrapping the application. Implementing the proposed approach ensures that the application is highly customisable and that the configuration is easily modifiable, even programmatically.

However, this is not enough; the configuration needs to have the possibility to target specific groups or specific users. For this purpose we propose a second structure, the target, which describes the parameters that need to match those of the front-end app making the request, ultimately the user/audience for a configuration. This can be any number of parameters with any value and some examples are: device header (brand, model, id, etc.), user profile information (age, gender, user-id, profile-id, etc.) or any custom information.

THE MACHINE LEARNING ENGINE

In essence we can formulate our problem as a graph generation and regression problem. We want to generate a configuration that can be represented as a graph and we want to find the best possible generated graph for the user, given their behaviours and additional user details. In other words, we have to find the best way to combine an algorithm that generates tree-structured graphs and an algorithm that is able to predict which generated graph is the best match for the user. We achieve this by calculating the television success rate of the generated configuration for that user's behaviours. Then we need to find an existing configuration that best matches the generated one. As a last step we have to modify existing configuration by merging it together with the generated config and the existing behavioural data of the user.

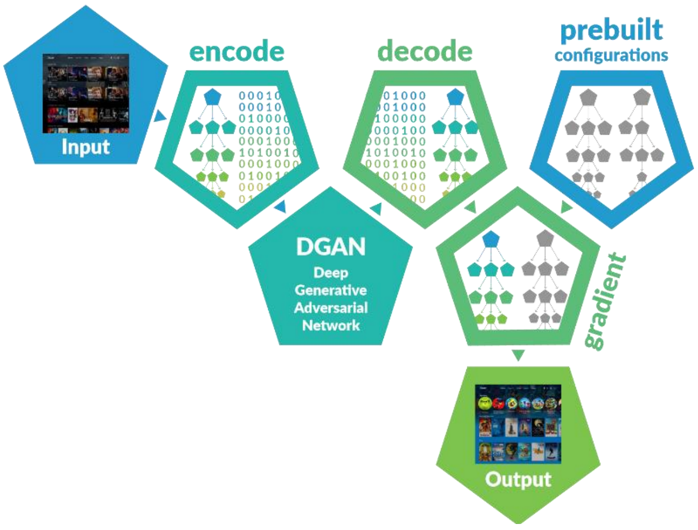

Figure 4 – The machine learning engine

The core of the machine learning engine, which we implemented for the purpose of this paper, is a *Deep Generative Adversarial Network (DGAN)* 'Goodfellow et al (16)' as shown in Figure 4. In a DGAN, we have two deep neural networks that compete with each other. We have the **Generator (G)**, which we use for generating new graphs, in essence new configurations. The other network is the **Discriminator (D)**, which classifies the generated graphs as valid or not. The two networks need to be trained together in order for the DGAN to work correctly.

We first decompose the JSON configuration to an adjacency matrix and a feature matrix, representing the type of node (page, module, element, etc.) and the sub-type of the node (page-type, module-type, element-type, etc.) as one-hot tensors. This process is called *Encoding* and the inverse process we call *Decoding*. After having our graphs encoded, we set up the neural networks *D* and *G*. In terms of the model and architecture of the deep neural network, this is a straightforward process. (Finding the best architecture of our network is subject to further study.) Then we continue by pre-training *D* to be an optimal classifier for the existing pre-created configurations, and a fixed *G*. Then, we stop the training of *D* and start to train *G* until *D* cannot differentiate between real data and generated data. This back and forth will lead to an optimal classifier and an optimal *G*, that can generate data very similar to the training set.

When we have the output from the DGAN, we still face another challenge - choosing the best configuration to modify. For the purposes of this section and for simplicity's sake, we treat them as UI designs created by professional UI designers for specific personas. 'Plinio et al (17)' The easiest solution would be to find the configuration closest to the output of the ML engine, and then modify that until it matches the output behaviour.

We also had to match the user's profile with the persona for which each individual configuration was designed by the UI designers. For example, if we have a persona called Casey (female 8-12), and we have a UI design created to match her expectations, that should be a very good starting point if the user *A* is a girl, age 9, who also happens to love *The Grinch*. But what if that layout is very far from what our machine learning engine outputs? This essentially means solving a system of non-linear equations. To tackle this problem, we used an optimised gradient descent algorithm. 'Ruder (18)'

Having a generated config, which is already a good one, matched with a pre-defined one and having the list of behaviours that the users have done, we need to modify the configuration to match the users' behaviours.

In the final step we already have a new UI configuration which is ideal for the users' needs, but in a real-world implementation we need to balance user needs with the business goals of the broadcaster or TV service provider. Therefore we have created an easy override function on top of the solution to make sure anything can be changed or tweaked to match these business goals.

CONCLUSIONS

We have demonstrated that it's possible to automatically tailor the digital television user interface to the needs each individual viewer. Using our solution as presented in Figure 6, we can split the solution into the following main parts:

- **UI** - Representing the front-end application that is based on a configuration
- **Configuration engine** - Representing the service that is able to create and serve targeted configuration
- **Analytics** - Representing the engine that is able to save and present the user behaviour data
- **Machine Learning** - The algorithms that recognise patterns, select and output possible raw configurations
- **Config generator** - Methods that convert and merge the output of the ML algorithms to create actual valid configurations for the app

We have introduced a mathematical formula to describe television experience success. We used machine learning to discover the behavioural patterns and predict an ideal UI to maximise the viewer experience. With this output, we managed to adapt the pre-created configurations to the needs of the individual viewer.

Overall, we think that machine learning and behavioural analysis will rewrite the rules of viewer experience in the next decade. This research constitutes just the beginning of what promises to be an exciting field of study.

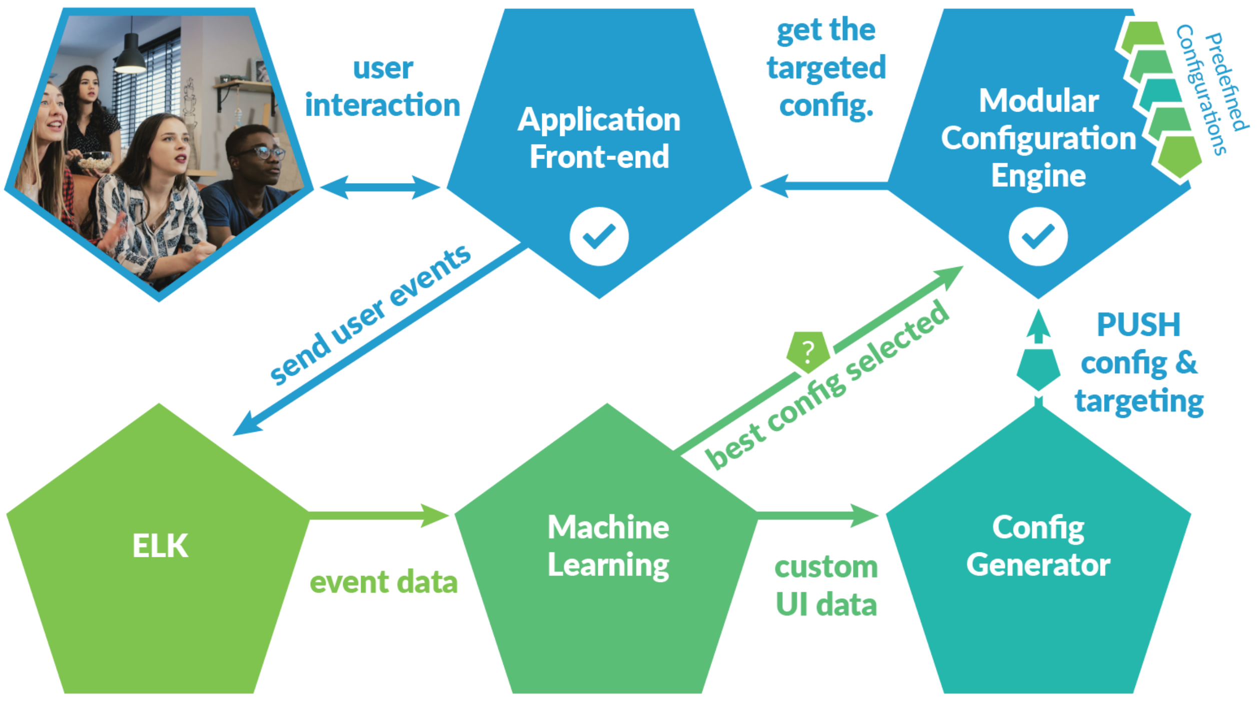

Figure 5 – Overview of the solution

REFERENCES

1. Joonhwan Kim, Younghwan Pan, and Brian McGrath, 2005. Personalization in digital television: Adaptation of pre-customized ui design.: 2nd European conference on interactive television: Enhancing the experience (EuroInteractiveTV 2005).

2. Young, I. Mental Models, 2008. Aligning Design Strategy with Human Behavior. Rosenfeld Media.

3. Szabo, Peter, 2017. User Experience Mapping. Packt Publishing.

4. Böhmer, Matthias and Krüger, Antonio, 2013. A study on icon arrangement by smartphone users. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 2137-2146.

5. Bernhaupt, Regina, et al., 2015. Tv interaction beyond the button press. IFIP Conference on Human-Computer Interaction. pp. 412-419.

6. Papp, I. I., Saric, Z. M. and Teslic, N. D., 2011. Hands-free voice communication with tv. IEEE Transactions on Consumer Electronics, Vol. 57, pp. 606–614.

7. Triolo, Cory, et al. 2018. Employing machine learning and artificial intelligence to generate user profiles based on user interface interactions. 15/610,701 US Patent, December 6, 2018.

8. Fogg, Brian J. 2009. A behavior model for persuasive design. Proceedings of the 4th international Conference on Persuasive Technology. p. 40.

9. Lichaw, Donna. 2016. The user's journey: storymapping products that people love. Rosenfeld Media.

10. Adomavicius, Gediminas and Tuzhilin, Alexander, 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge & Data Engineering, Vol. 17, pp. 734-749.

11. Basu, Chumki, Hirsh, Haym and Cohen, William, 1998. Recommendation as classification: Using social and content-based information in recommendation. Aaai/iaai. pp. 714-720.

12. V Vijayakumar, Subramaniyaswamy Vairavasundaram, R Logesh, and A Sivapathi, 2019. Effective knowledge based recommender system for tailored multiple point of interest recommendation. International Journal of Web Portals (IJWP). Vol. 11, 1, pp. 1–18.

13. Henaff, Mikael, Bruna, Joan and LeCun, Yann, 2016. Deep convolutional networks on graph-structured data. CoRR, abs/1506.05163, 2015.

14. Frost, Brad 2016. Atomic Design. Brad Frost Web.

15. Bray, T. 2017. The javascript object notation (json) data interchange format. STD 90, RFC Editor.

16. Goodfellow, Ian, et al. 2014. Generative adversarial nets. Advances in neural information processing systems, pp. 2672–2680.

17. Filgueiras, Plinio Thomaz Aquino Junior and Lucia Vilela Leite, 2005. User modeling with personas. Proceedings of the 2005 Latin American conference on Human-computer interaction. pp. 277–282.

18. Ruder, Sebastian, 2016. An overview of gradient descent optimization algorithms. CoRR, abs/1609.04747.