Aalto-yliopisto
Aalto-universitetet
Aalto University

ELEC-E7852

Computational Interaction and Design

Assignment A2a

Bayesian optimization

Aitor Urruticoechea Puig
aitor.urruticoecheapuig@aalto.fi
Student N°101444219
November 2024

Notice

Acknowledgement

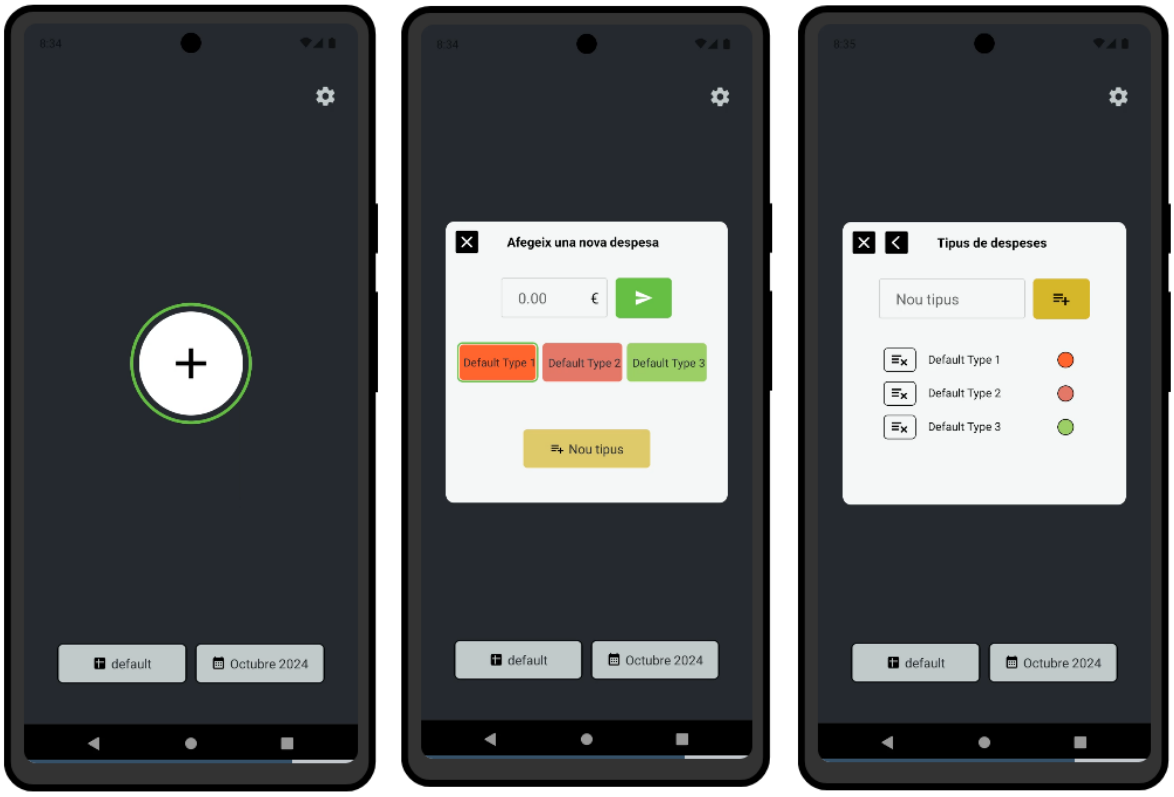
Parts of the code implemented in this work were generated with the assistance of GitHub Copilot. None of its proposals were, however, employed without refinement and necessary tweaks to adapt it to the nuances of the tasks at hand; making it impossible to identify and subsequently mark which lines of code were human or machine-made, for many were the fruit of the combination of both.

Contents

List of Figures

1 Introduction and case description

From "Urruticoechea", the assigned topic from the assignment A2a is "Optimizing a design with a designer in the loop (human-in-the-loop design)". Just as in the first assignment, a side project that I have been slowly developing is to be used as a playground. This is a spending tracking app which should help synchronize all your spendings and update your custom Google Sheets file with ease. Its UI is very much still in development, but can be seen in Figure 1 (unfortunately, the UI is only available in Catalan at the moment). The idea is to simplify the process as much as possible. The user needs only to choose a spending type and add the amount, the system should do the rest (Figure 1b). Spending types can be added, removed, and edited in a separate screen (Figure 1c).



(a) Home screen (b) "Add new spending" window. (c) "Spending types" window.

Figure 1: "DespesApp" spending tracking app overview.

In this case, the objective is to iteratively refine the app's user interface by incorporating feedback from a designer at each step. The goal is to achieve an optimized design that balances user needs with aesthetic and functional preferences defined by the designer. This will be specially helpful for the functionality of adding spending types the add has (Figure 1c). This window needs to allow for different actions to be performed in the same place: adding and editing spending type names mainly; but also removing them, assigning them colours, and re-ordering them according to the user wishes. This leads to a large enough design space with many moving parts that can clearly benefit from this type of iterative designer input.

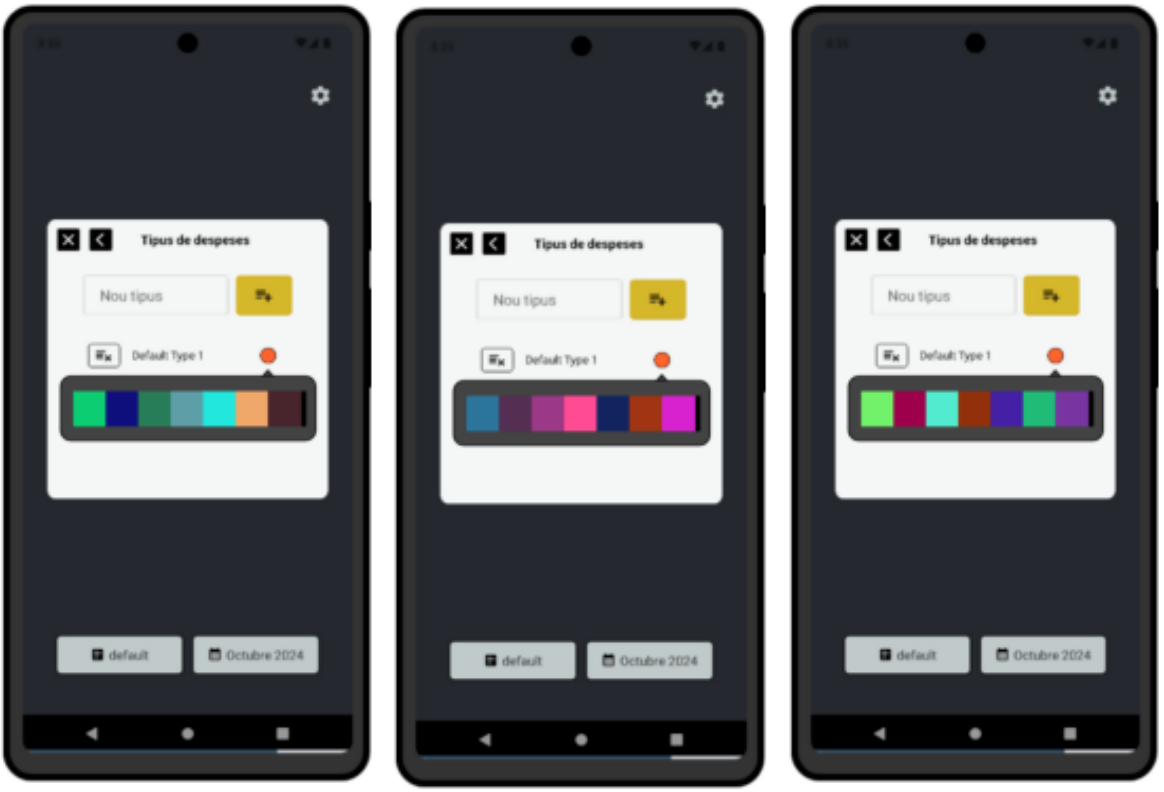
This makes even more sense when integrating Bayesian Optimization into the iterative process, as it works best in this kind of complex design spaces without needing too many feedback iterations necessarily. Basically, because this allows the designer to guide the optimization process without needing to evaluate every possible design. For an assignment like this, however, let us focus on a more concrete issue so we can iterate on one concrete design parameter. In this case, the RGB

colour picker to choose the colour for each spending type. In this way, different proposed designs can be presented to the designer with only "one" variable change: the RGB colour picker, and optimization gets simplified to better reflect what has been taught in class.

2 Bayesian Optimization Implementation

The main philosophy behind the implemented Bayesian Optimization is, as explained before, to repeatedly show the designer options for a colour picker; with the final goal of finding a colour distribution the designer would rate highest. Crucially, extra effort has been put to show the potential design outputs with relevant context for the designer, rather than just the colour picker bar in a vacuum.

The algorithm will start by generating 5 random colour distributions that the designer will rate before the optimization loop begins (as an example, see Figure 2a). This initial random distributions serve as a starting point for the optimization loop, where `SingleTaskGP` is used to create a Gaussian process model. For it, the tuples defining the colour bar are flattened into `TrainX` and the ratings are fed as `TrainY`. This is iteratively optimized using a rather typical Adam optimizer. After the optimization step is performed, from `BoTorch`, the Logarithmic version of `ExpectedImprovement` is used to define the acquisition function (as it was shown to perform better than the linear version). The acquisition serves as a base for the generation of the next sample which is shown to the designer, re-starting the loop all over again. When the iteration limit is reached, a last sample is generated with what is believed to be the best possible RGB colour picker (see Figure 2c).

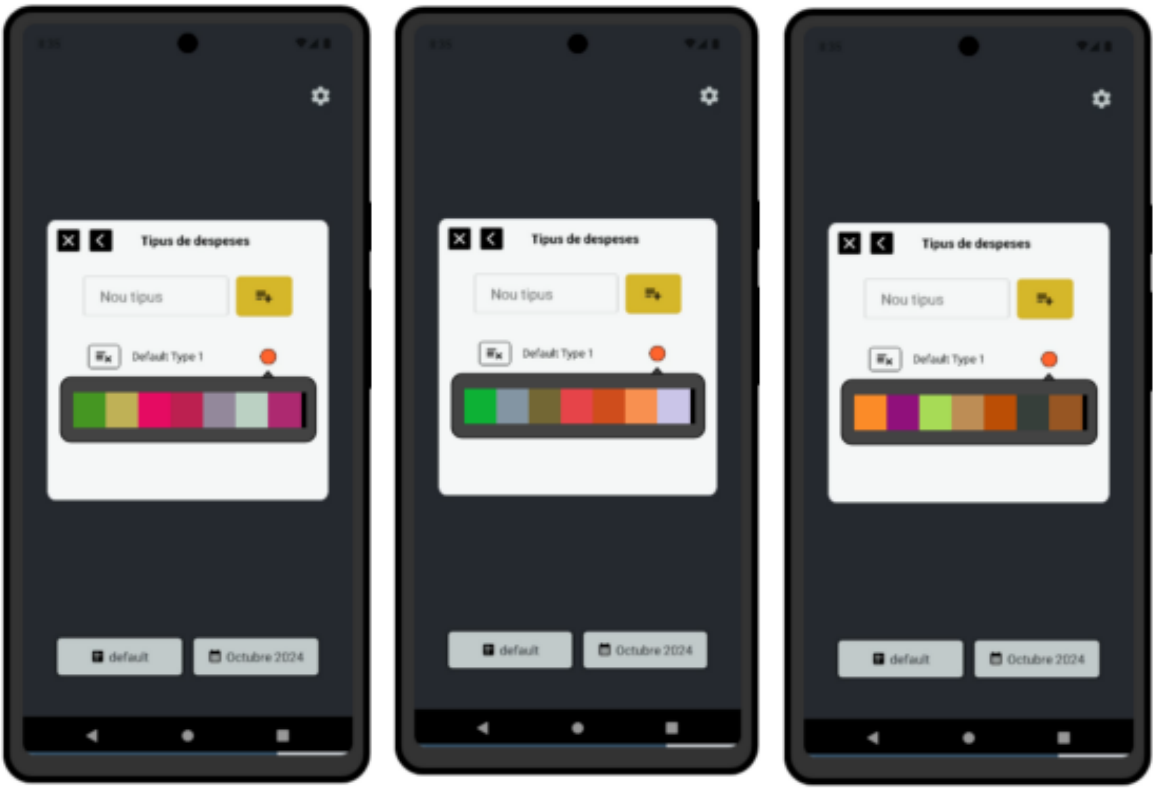


(a) Randomized start. (b) Exploration sample. (c) Result after 10 iterations.

Figure 2: Optimization process for the RGB colour picker.

3 Results

Figure 3 includes a sample of three obtained optimized results after 10 iterations. To obtain them, the input given has focused on the app context and what it needs, rather than what would be an optimal colour picker for any use; which is the main point for this exercise at the end of the day. While the results are satisfactory enough, and very much in-line with what I would, as a designer, expect to find for colours to label spending types; the algorithm is far from perfect. Namely, because the balance between exploration and exploitation still has room for improvement. It basically does prioritize exploration while it fails to exploit promising improvement paths (i.e. too many samples rated with 1, not enough with 4). Of course, this does not necessarily mean to bad results, as due to the share amount of sample points the optimization algorithm is able to indeed fit a function that results in results with high enough rating, but this does not deny the fact that improvements could be done to ensure better results with the same sample number.



(a) Optimized result 1. (b) Optimized result 2. (c) Optimized result 3.

Figure 3: Optimized results sample.

4 Conclusion

The usage of this type of Bayesian Optimization for this case study is very much called for. In this case, where the goal is to use a colour picker for potential types of spending, it does not necessarily make sense to use a standard-perfect RGB colour picker. Instead, optimizing with a designer in the loop makes more sense: the goal is to find colours that would be relevant for types of cash spending while maintaining certain levels of colour coherence with the rest of the app. The proposed algorithm is able to iterate on that, though suboptimally when analysing its exploitation capabilities, and return satisfactory results that are contextually relevant and useful.