Human Computer Interaction

Victor Møller Poulsen, Studie Nr.: 201707639

June 6, 2021

1 Abstract

This paper presents a prototype Streamlit app which teaches a Gelman (Gelman et al., 2020) inspired Bayesian workflow in R with the package brms (Bürkner, 2017) and in Python with the package pyMC3 (Salvatier et al., 2016). The app is built with theories from user experience (UX) in mind and relies on a well-documented codebase. The app is aimed at users who are already familiar with Bayesian analysis in either R (brms) or Python (pyMC3) and should be used as an introduction to the implementation that the user is unfamiliar with. The app has several interactive elements, including optional sections that the user can expand to dive deeper into parts of the material, quizzes with feedback and live rendering of plots and figures based on which model and priors the user is interested in.

Link to app:

Link to Github:

2 Introduction

This paper is a presentation of a prototype Streamlit app which teaches the user a good Bayesian workflow in R/brms and in Python/pyMC3. It relies primarily on methods from user experience (UX) and data science. The main idea is to teach users who know how to conduct a Bayesian analysis in one coding language how to conduct a similar analysis in the other coding language. This is something that I had to figure out when we transitioned from using R and brms for Bayesian analyses on the Cognitive Science bachelor degree to using Python and pyMC3 on the Cognitive Science master degree. There are lots of great resources and examples online and in books for both brms and pyMC3 independently, but I have so far not found any sources that directly map between the two. That is the niche that this Streamlit app is intended to occupy. Besides differing in content from books and online resources, this app is also intended to be interactive. This is important because it facilitates user engagement and agency. It also useful because it means that we can let the user select what they want to see on their screen and which parts they would like to engage more with. It thus saves a lot of clutter from the app, because we do not have to display everything at once. This is where Streamlit becomes relevant. Streamlit is a Python package which allow users to launch interactive web applications to showcase analyses. It thus fulfills the same function that Shiny fulfills in the coding language R. Streamlit is developing rapidly however, and streamlit already appears to support more diverse options. This is the reason why Streamlit was used rather than Shiny in this case.

3 Target Group

The target group for this app is people who are familiar with Bayesian statistics and know how to conduct a Bayesian analysis in either R (brms) or Python (pyMC3). The idea is to conduct (almost) identical analyses in the two languages, such that users can map what they already know in one language on to the other language. Explanations are targeted more towards users who want to transition from R (brms) to Python (pyMC3) than the other way. This is because I have kept cognitive science students in mind as the primary audience. Python and R are the two largest Data Science programming languages, and as such it is common for Data Scientists to code in both languages. Thus, the app should be useful for users outside of Cognitive Science as well.

4 Intended Usage

The first use case (UC1) that I think is reasonable is to use the app as a one-time thorough introduction, where users should ideally run their own code in Python and/or R while following the app. This should be relatively easy since I provide reproducible code for all parts of the analysis, which can be easily copied to clipboard from the expandable code-chunks. For this use case the user will have to interact with the page, expand code-selections and explanations and tweak the plots to get a feel of what is going on. The user might also learn something new, either conceptually or related to a healthy workflow. This however is secondary to the main goal of building a bridge between the code implementations. The second use case (UC2) that I think makes sense is to use the app repeatedly as a reference while conducting an analysis in either pyMC3 or brms. The user can quickly glance through the app to be reminded of how a good Bayesian workflow can look for either implementation. I have also included a section with references that I have found helpful for both pyMC3 and brms, and as such the user can also use the app as a gateway to more advanced analyses. For this use case the user might not want to expand the optional sections, but simply glance through the plots and headers to quickly find what they are looking for.

5 Design Philosophy

The design philosophy for the app is naturally related to the content that it attempts to convey and introduce to the user. It is also designed to facilitate both UC1 and UC2. The design philosophy burrows heavily from Hasselzahl's (2003) Pragmatic/Hedonic model of User Experience (UX) as well as the Macintosh Human Interface Guidelines (1995). Firstly, the app is designed with a philosophy of *avoiding clutter* in mind. I have hidden everything that I think is not necessary for navigating the app and using it in line with UC2 in expandable sections. I wanted to avoid cluttering the app with text specifically since this can feel intimidating and could scare users away. If users want to use the app in line with UC1 they will have to interact with the optional elements and decide to see the code-chunks and the optional explanatory sections. If these were shown by default, I think that UC2 would not be facilitated well by the app. Secondly, the app is designed with *consistency* in mind. This importance of consistency is discussed in the Macintosh Human Interface Guidelines (1995) where they highlight that consistency allows users to transfer knowledge from one application to another. In our case, since we only have one app, what will be important is that users can transfer what they have learned in one section to other sections. This is what has been referred to as "internal consistency" (Macintosh Human Interface Guidelines, 1995, p.) The users will quickly learn that there are three types of expandable boxes in the app: "Code-Monkey", "Language-Learner" and "Concept-Guru". Each of these expandable sections allow the user to access different types of additional info. When clicked, the "Code-Monkey" sections will pop out and show reproducible code, "Language-Learner" sections go into more detail about how R/brms differ from Python/pyMC3 while "Concept-Guru" sections discuss Bayesian analysis more generally. Each subsection starts with a header and is then followed by a brief text. Below this are plots or outputs from Python and R, and only after this will the user find the optional sections. These are also consistently in order, such that "Code-Monkey" comes first, "Language-Learner" second and "Concept-Guru" lastly. Additionally, each expandable section has a unique icon associated with it. This has been implemented to make it easier for the user to navigate the page. The Macintosh Human Interface Guidelines (1995) focus heavily on the importance of icons. One highlighted advantage of icons is that users typically will recognize images faster than they will read text (Macintosh Human Interface Guidelines, 1995, p. 224). Third, the app should be *pretty* and *simple* (plots, color theme). I have created a custom color-theme for the app, which I think makes the app more interesting than with the standard white theme. I have also tried to make the formatting both inside streamlit, and for the R and Python plots pretty. Considerations around making the plot engaging and pretty, is another reason why I show plots by default and hide most text and code by default. While this does not directly relate to either functionality or usability, which are key design concepts, pleasure is also understood to be an important driver in creating an enjoyable experience (Hasselzahl, 2010, p. 65). I have found generally found that having a clear target group in mind makes it easier to design an app with high usability. We can facilitate the specific type of interaction that we want the user to have. If the target group and intended usage is too diverse it can lead to clutter.

6 Interactivity

There are three basic types of interactivity implemented in the app at this point. Firstly, the user can choose which model and prior to see analysis for. Plots and figures from R and Python are shown by default and cannot be toggled off by the user. However, the user can influence what is shown in plots. In most cases the user can choose among different prior options that have been used to fit the models in the app. In some cases, the user can choose to plot the outputs of analysis between different models, e.g. with different random effects structure and likelihood functions. When the user makes a selection the plots and figures will change, and the code in the “Code-Monkey” sections will also change in the cases where the analyses differ. Secondly, the user can choose to expand the optional sections: “Code-Monkey”, “Language-Learner” and “Concept-Guru”. As I have already said, the “Code-Monkey” section will render code which matches the selection. Most often, the content in “Language-Learner” and “Concept-Guru” will not differ based on the selection of the user, because these sections contain more general considerations. Lastly, the user can take quizzes and receive feedback. These quizzes are not shown on the page by default but hide inside the expandable boxes. This is again to ensure that the app will not be too cluttered and can be glanced through quickly (for UC2). Based on the answers that users give, they will receive feedback on whether they have the right intuitions (dialogue)? Streamlit is often used for Data Science applications which run extremely fast machine learning analyses or frequentist statistics. The apps are typically interactive in the sense that when the user makes a choice some corresponding code is run, which generates an output. This means that the potential functionality is almost unlimited, i.e. there are often infinitely many (or very many) ways of combining different choices to receive specific outputs. This is not the case in the prototype app that I am presenting. In my case, I am not running any analysis code live because full Bayesian sampling is prohibitively slow. Thus, I have made an extensive code base from which I generate plots and outputs from several analyses (e.g. for different priors and different models) that the user then sees based on selection. As such the functionality and interactivity is limited to what I have already pre-run. This is a difficult trade-off (functionality vs. usability), but I believe that the app should be engaging, and if users really want to tweak the analysis, they *should* run their own code anyways. After all, the whole purpose is to teach users how to implement Bayesian analysis, and the best way to learn that is to do it yourself.

7 Code Base

Although I have focused extensively on the user experience (UX) on the app, the most demanding work has been to develop good analyses and managing an extensive code base which must be reproducible. I will not go too deep into that here, as you can check the Github (link). There are bash-scripts for both R and Python which run all the necessary Python and R scripts to run the whole pipeline and reproduce all analysis on the streamlit app. This means that if users have the necessary packages in their environment, they can run all R and Python analysis with two lines of code from their terminal. This is of course unnecessary for most users, who should just see the app, but it is an option for users who want to dive deeper into the analysis and the convenience functions that I have created. Besides this, a well-structured code base has been absolutely necessary for me to be able to manage the project and ensure that all formatting is consistent.

8 Improvements & Testing

COLOR-CODE Something I really would have liked to implement in the app is a consistent color-code for the expandable sections that I have. This would make sense because I use the same three types of boxes ("Code-Monkey", "Language-Learner" and "Concept-Guru") throughout. A color scheme, where each type of section has its own color-code should make the page more intuitive to navigate. Unfortunately, this functionality is not natively supported in Streamlit. Streamlit can be configured with CSS code, so in principle it should be possible to implement. Unfortunately, I am unfamiliar with CSS so there is currently no color-coding for the expandable sections. As I have mentioned previously, each section has an associated icon, which aims to achieve the same. I do think that colored sections would make it more visceral and would improve the user experience.

FEDBACK & INTERACTIVITY With regards to feedback and interactivity there is one thing that would be ideal to implement for the use cases that the app attempts to facilitate. At present, the user is only quizzed with multiple choice questions. This is because it is easy to handle, since I can specify the feedback for a limited number of options. Because the app attempts to teach the user code-practices it would make sense to have a system which corresponds to what is used on platforms such as DataCamp. On DataCamp users type in code, which is then evaluated, and the user received feedback based on the code. The tricky part is that the code has to be compiled, and the app has to provide feedback on some test of the code. The tricky part is that code which does the same can be written in infinitely many ways,

so it is not possible simply to take the input as a string and evaluate that. There also has to be a nice interface where the user can type in the code that they want to execute. I think that both would be tricky to implement, especially the first part. Another reason to not support this functionality is that I would ideally like users to code along from their own instance of R and/or Python.
TESTING I intend to share the app in cognitive science forums once I have double checked that everything is reproducible, and that everything is well documented. Hopefully, the app will be useful to some of the students, and perhaps some outsiders will even find their way to the app. Additionally, sharing the app should give me the chance to receive feedback on whether the app is usable. Both whether the content is good and whether the format is intuitive. I intend to just gather verbal feedback, although I know that this is not the ideal testing conditions. In general, it would be nice to have made two apps with different layout and A-B test them against each other. While I did work on this idea at an early stage, it just turned out to be too big of a project unfortunately. It would also be interesting to gather physiological data (e.g. eye-tracking data) or simply gather footage of users interacting with the app (e.g. screencast and mouse-tracking). What I am really interested in is *how* people interact with the app (e.g. UC1, UC2 or something else). It would be useful to observe whether people interact with the app, expand the boxes, try different priors, etc.

9 Conclusion

This paper has introduced a streamlit app which teaches a Bayesian workflow in R (brms) and Python (pyMC3). The app is designed for users who are already familiar with Bayesian analysis in either R (brms) or Python (pyMC3). The app focuses on building a bridge between the implementations in the different languages and packages. The app supports limited interaction through selection boxes, and expandable sections for the user who wants to dive deeper. The app also attempts to engage the user with quizzes and provides feedback based on answers. The app is designed with a philosophy of consistency, simplicity and beaty in mind. Users who want to go further than the app supports can consult a well organized code-base at Github (link).

References

Bürkner, P.-C. (2017). brms: An R package for Bayesian multilevel models using Stan. *Journal of Statistical Software*, *80*(1), 1–28. https://doi.org/10.18637/jss.v080.i01

Gelman, A., Vehtari, A., Simpson, D., Margossian, C. C., Carpenter, B., Yao, Y., Kennedy, L., Gabry, J., Bürkner, P.-C., & Modrák, M. (2020). Bayesian workflow. *arXiv preprint arXiv:2011.01808*.

Salvatier, J., Wiecki, T. V., & Fonnesbeck, C. (2016). Probabilistic programming in python using PyMC3. *PeerJ Computer Science*, *2*, e55. https://doi.org/10.7717/peerj-cs.55