Modelling Human Behavior in HCI:
Transition from menus to keyboard shortcuts.

Teacher:

- Gilles Bailly (gilles.bailly@sorbonne-universite.fr)

Requirements:

- Python programming
- HCI basics
- Mathematic basics (probability, log, exp, etc.)

Libraries

- Tested with python 3.7, seaborn 0.11, numpy 1.18.5, matplotlib 3.2.2

Objective

Develop and test models to predict and explain human behavior. The application case is the transition from menus to keyboard shortcuts.

Compétences acquises à la fin du TP :

- Use simple reinforcement learning models often used in decision making and neurosciences
- Develop models for sequential decision making
- Adjust the parameters of a model
- Simulate a model
- Compare models with the tools log-likelihood and BIC score

Information

- For each question, you have an estimate of the time to complete it.
- For a step of the Exercise **i**, you will find one or more comments in the corresponding code of the form **# TODO Step i**

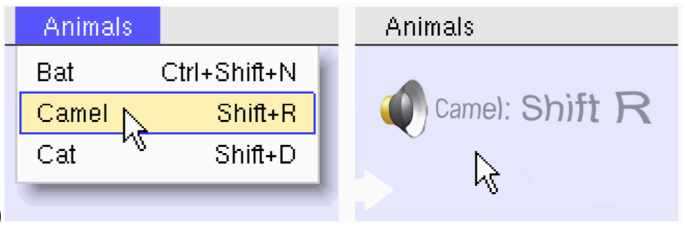
Before the exercise

- Read pages 1, 2, 3 (theoretical part)
- Read the theoretical boxes (definition of models, definition of metrics)
- Do step 1 (5mn) to check that you have the required libraries.

HCI context

Keyboard shortcuts (or Hotkeys)

How can users be encouraged to use the effective methods at their disposal to accomplish a task? Typically, keyboard shortcuts, e.g. Ctrl+S for the Save command, are particularly effective. Unfortunately, they are seldom used: many users, even experts, continue to use menus rather than make the transition to keyboard shortcuts.
Many interaction techniques are regularly proposed to promote the use of keyboard shortcuts [Bailly et al. 2016]. For example, one interaction technique plays (audio feedback) the name of the keyboard shortcut when the command is selected in the menu (Figure 1).



)

Figure 1: Interaction technique playing the keyboard shortcut when a command is executed from the menu [Grossman et al. 2007]

Data

In [Grossman et al. 2007], the authors conducted an experimental study to compare several interaction techniques to promote the use of keyboard shortcuts (Figure 2). The authors compare three techniques (Traditional, Audio, Disable). In this Exercise, we consider Traditional (simple presentation of the shortcut to the right of the item) and Audio (Figure 1) techniques. Each participant tests only one technique and has to execute about 720 commands (each command has different frequencies. Some commands appear frequently, others rarely).

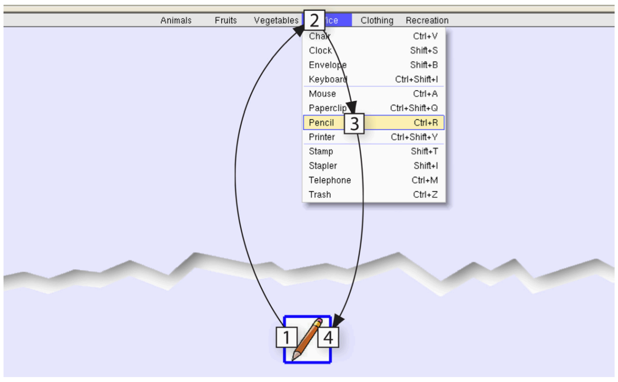


Figure 2: Un essai : Une image est présentée en bas de l'écran. Le participant doit exécuter la commande correspondante à l'aide du menu ou, s'il le connait, du raccourci clavier correspondant.

At each attempt, the target command appears on the screen and the user executes it either with the MENU method or with the HOTKEY method. But, from a cognitive point of view, we consider that the user chooses one of these three strategies:

- MENU: I select the command with the MENU method.
- HOTKEY: I select the command with the Keyboard shortcuts method.
- LEARNING: I learn the keyboard shortcut by executing the command in the menu.

For each command execution, the time and whether the command was executed correctly or not is also recorded. For more information :

- Gilles Bailly, Eric Lecolinet, and Laurence Nigay. 2016. Visual Menu Techniques. ACM Comput. Surv. 49, 4, Article 60 (February 2017), 41 pages. https://doi.org/10.1145/3002171
- Tovi Grossman, Pierre Dragicevic, and Ravin Balakrishnan. 2007. Strategies for accelerating on-line learning of hotkeys. ACM CHI '07. 1591–1600. https://doi.org/10.1145/1240624.1240865

Approach: Reinforcement Learning (RL)

Figure 3 illustrates the principle of reinforcement learning. Given a **State** s, the agent performs an **Action** a, receives a **Reward** r and is in a new **State** s' = s_{t+1}.

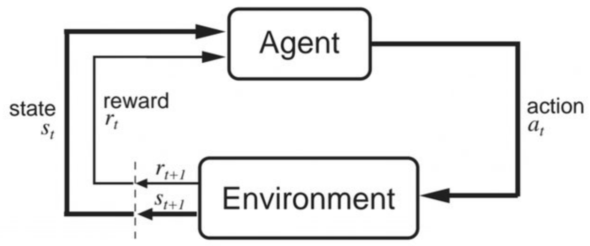


Figure 3: Reinforcement Learning

We formulate the problem of the transition from menus to keyboard shortcuts as a Reinforcement Learning problem where the interface is the environment and the user is the agent. Given a command s to execute (State), the agent chooses a strategy (Action) among MENU, HOTKEY, LEARNING. The reward (here cost), is the time it takes the agent to execute the command (to simplify we make the assumption that the user does not make a mistake).

Exercise

This TP is in three main parts:

1. Given a model, its parameters, and participant data, estimate the model's likelihood, i.e., how well the model fit the participant's data.
2. Optimizing the models, i.e. finding the optimal parameters of the model that maximizes the likelihood function.
3. However, likelihood is often not sufficient to ensure that these models accurately reflect user behaviour. It is necessary to perform simulations and to verify that the data produced are compatible with observed human data.

Two models are given to familiarize you with the theoretical concepts. You will implement two other models. Depending on how much time you have left, you will be given several directions to create your own models.

TP

1. Introduction (5mn)

1.a Download the archive <xxx>.

Run the file that loads and displays user data.

> python ./main.py

We see for each participant, all command executions (720) with selection time (y axis) and the strategy used (color). The title of the figure also indicates the interaction technique.

Q0: What is the role of the interaction technique on the strategies used?

Q1: What is the problem with this visualization?

1.b Visualisation per command

Uncomment the lines corresponding to this question to solve the previous problem. This new visualization now allows you to see what happens for each command. It allows you to identify frequent and less frequent commands. We will use these visualizations to debug the models if needed.

Q2: Are users making more use of keyboard shortcuts for frequent commands?

We want to model these sequences of strategies for each user and each command. We will start with two models: Radom and Choice Kernel.

2. The « Random » and « Choice Kernel » models (5mn)

Random pattern. This model chooses a random strategy with a bias for one of the options. In our case, the bias is captured with the parameter HOTKEY_PROB which is between 0 and 1 such that the probability of using the strategy a for the command c:

P(c, a) = HOTKEY_PROB si a == HOTKEY
P(c, a) = (1 – HOTKEY_PROB) / 2 si a != HOTKEY

Choice Kernel (CK) model. This model, often used in neuroscience, captures the user's tendency to repeat previous actions. The user calculates a "choice kernel" $CK_t^k(c)$ that remembers the frequency of use of the strategy (or action) k for the command c .

$$CK_{t+1}^k(c) = CK_t^k(c) + \alpha_{ck}(a_t^k - CK_t^k(c))$$

where $a_t^k = 1$ if strategy k is used in trial t, otherwise $a_t^k = 0$ and α_{ck} is the learning rate. To simplify, we consider that the initial value of the kernel choice is 0. We can then calculate the probability of executing the strategy a for the command c using the softmax function:

$$P(c, a^k) = \frac{\exp\left(\beta_{ck}\, CK_t^k(c)\right)}{\sum_{i=1}^K \exp\left(\beta_{ck}\, CK_t^i(c)\right)}$$

where β_{ck} is the inverse temperature which is the level of stochasticity. $\beta_{ck} = 0$ means that the choice is random. $\beta_{ck} = \infty$ means the chosen option is the one with the largest value. This model has two parameters $(\alpha_{ck}, \beta_{ck})$.

Comment the code corresponding to the steps 1.a et 1.b to save time

2.a Look at the files in /plugin/model/.

For each model, the main methods to reimplement are :

- **action_probs(self, cmd)** which given a command to execute (a state in the RL formalism) returns the probability that the agent chooses each strategy (action in the RL formalism).
- **time(..)** and **success(..)** for the reward (you can ignore)
- **update_model()** which updates latent variables (e.g $CK_t^k(c)$).

The template parameters are in ./parameters/. The column freedom indicates if the parameter is "free", i.e. if it can be varied (=1) or if it is fixed (=0). You can see that there are several fixed parameters for time estimation because this is not the focus of this exercise.

For this exercise, the random and CK models are already implemented. Look at the code.

3. Estimate the likelihood of the models (25mn)

Given a model and its parameters, how well did the model fit the participant's data?

The **likelihood function** reflects the model's ability to replicate a participant's action choices for each trial. From a Bayesian perspective, it is the maximum probability that model *m* will choose the same set of actions as the participant. This consists in analyzing the model prediction conditional on a history. That is, estimating the likelihood that participant *p* will choose the action a_t^k at the trial *t* given the history ($d_{1:t-1}$) of the participant from the trial *1* to the trial *t-1*. Formally,

$$ll(m,p) = \sum_t \log P\left(a_t^k \middle| S_{1:t-1}^p, m, \theta_m^p\right)$$

where *m* is the model, *p* is the participant, θ_m^p is the parameeters of the model *m* for the participant *p* and $S_{1:t-1}^p$is the sequence of actions performed by the participant *p*.

However, a limitation of this function is that it favours "complex" models, i.e. models with many parameters. For this, the Bayesian Information Criterion (BIC) score is often used to penalize models with many parameters:

$$BIC = ln(n) * k - 2 * ll(m,p)$$

n the number of observations to be predicted, *k* the number of model parameters and *ll(m,p)* the model likelihood.

We want to calculate the likelihood for each participant and each model. Caution! To analyze the models more precisely, we will keep for each trial the probability of choosing the participant's action.

3.a Implement the method **run_debug**() of the class **Individual_Model_Fitting** in ./lib/model_fit.py (10-15mn)

This method takes as input:

- The sequence of commands to be executed for a participant (command_sequence)
- An object of type User_Output (see util.py) which contains three lists: time, success and action, describing how the participant executed each command.

The method output is an object of type **Fit_Output_Debug** which contains mainly a list (probs) containing the probability for each trial that the model will choose the user action. Ignore for the moment the output attribute.

The objective is to update the **Fit_output_Debug** structure to contain the probability that the model returns the action performed by the participant. To do this, we go through the sequence of commands to be executed and calculate the probability that the model will choose the user action. Do not forget to update the model using the method **model_update**() with what the participant has actually done.

3.b Implement **log_likelihood()** and **bic_score()** in util.py (2mn)

When necessary, use the numpy functions.

3.c Print the results (5mn)

- Estimate the likelihood of the model using the code given in main.py
- Display the results using the code given in main.py
- Save the graphs ("Save" button) for later.
- Compare the models.

Q3 What is the best model? For which users? For which techniques? Must the bars be high or low? Look at the label of the y axis as well as the sign of the likelihood.

However, the parameters of the models have been chosen arbitrarily. You can manually test other parameter values by editing the corresponding files in /parameters/. You can also automatically search for the optimal parameters (step 4).

4. Optimiser les paramètres des modèles (15mn)

We use an optimization method to explore the parameter space of the model and return the parameters that maximize the likelihood function. This likelihood function is called in the **to_minimize()** method. Many optimization methods exist.

4.a Modify the method **optimize()** in model_fit.py (5mn)

We will use

- the **differential_evolution()** method from scipy.optimize (model_fit.py)
- the method to minimize is **to_minimize**() (model_fit.py)

4.b Launch the optimization method (2mn)

Execute the corresponding code in main.py. Please note that the optimization may take some time (on my computer, about 1mn). If it happens, the results of the optimization are already in the folder: ./optimal_parameters_copy/

4.c Visualize the likelihood with optimal parameters (5mn)

The optimization results have been saved in /optimal_parameters/ (in case of problems you have a copy in /optimal_parameters_copy/).

- Comment in main.py the code corresponding to the optimization (4.b)
- Modify in main.py the code to display the likelihood (3.b) in order to take into account the optimal parameters (instead of the default parameters). To do so, add this line:

model_fitting.parameters = Parameters_Loader.load('./optimal_parameters/')

Q4 Does the likelihood change? In absolutely and in relative? What is now the best model? for which users? for which techniques?

5. Model simulation (20mn)

We will now simulate the model and see if the data produced "freely by the model" reflects the user's data. Here, we are not looking at the probability that the model chooses the user's action, but at the sequence of actions freely produced by the model (two simulations with the same model/parameters can therefore produce different sequences).
To evaluate the sequence of actions produced, we will use a common metric for the study of keyboard shortcuts: the percentage of use of keyboard shortcuts according to practice.

5.a Implement the method **simulate()** in model_simulation.py (10mn)

You can get inspiration from the method you implemented in the model_fit.py file. But beware, here the model is freewheeling, i.e., it does not use the participant data. The model updates itself according to the actions it generated. You will use the methods of the class Model_Interface:

- **select_strategy()** (model_interface.py)
- **generate_step()**
- **update_model()**

5.b Execute and visualize the simulations (10mn)

Use the code provided in main-exercice.py. Analyze the figures illustrating the evolution of keyboard shortcuts over time.
Q5 What is the "best model"?
Q6 Do the models look good in absolute terms?

6 Implement the Rescorla-Wagner model (10mn)

It should be noted that previous models did not use rewards. This is now the case with the Rescorla-Wagner model.

Rescola-Wagner Model (RW). This model learns the $Q_t^k(c)$ *value* associated with each strategy *k* and each command *c* based on the history of the actions and the associated gains. The model then uses these values to decide which action to perform :

$$Q_{t+1}^k(c) = Q_t^k(c) + \alpha_{rw}(r_t - Q_t^k(c))$$

where r_t is the reward and α_{rw} is the learning rate between 0 and 1. We can then estimate the probability to choose a^k for the command c according to the softmax function as the Choice Kernel model:

$$P(c, a^k) = \frac{\exp(\beta_{rw}\, Q_t^k(c))}{\sum_{i=1}^{K} \exp(\beta_{rw}\, Q_t^i(c)}$$

where β_{rw} is the inverse temperature defining the level of stochasticity. This model has two parameters (α_{ck}, β_{ck}).

- Implement this model. A template is given in ./plugins/models/
- Estimate the optimal parameters for this model or use the ones in /optimal_parameters_copy/
- Estimate and visualize the results of the likelihood function and simulations

Q6 What do we learn ? why ?

7 We can go further (10mn)

It is possible to combine these models. In neuroscience, it is common to combine Rescorla Wagner and Choice Kernel (RWCK). You can implement this model.

Rescola-Wagner + Choice Kernel (RWCK). This model mixes the two previous models. The previous equations will be used to update the internal variables. The probability of choosing an action is now defined as :

$$P(c, a^k) = \frac{\exp(\beta_{rw}\, Q_t^k(c) + \beta_{ck}\, CK_t^k(c))}{\sum_{i=1}^{K} \exp(\beta_{rw}\, Q_t^i(c) + \beta_{ck} CK_t^i(c))}$$

This model has four **parameters** (α_{rw}, β_{rw}, α_{ck}, β_{ck}).

Q7 What do we learn ?

7 We can go even further

It is possible to implement other models with different mechanisms.

- Explicit learning: Each time the user uses the LEARNING strategy for a c command, he learns the keyboard shortcut and thus increases the $Q_t^{HOTKEY}(c)$"value" with a learning rate α_explicit
- Implicit learning. Each time a menu selection is made, the user implicitly learns the keyboard shortcut. The value $Q_t^{HOTKEY}(c)$) would increase with a learning rate $\alpha_{implicit}$ such as $\alpha_{implicit} \ll \alpha_{explicit}$
- A mechanism of oblivion. At each try, the user forgets (a little) all the shortcuts in memory with a decay rate.
- A mechanism you would like to test.

Combinations that work well:

- Explicit learning + Decay + Choice Kernel
- Explicit learning + Decay + Implicit learning