TP Modélisation du comportement humain en IHM : Transition des menus vers les raccourcis claviers.

Encadrant :

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

Prérequis :

- Programmation python
- Initiation à l'IHM
- Bases en mathématiques (probabilité, fonction logarithmique, fonction exponentielle, etc.)

Librairies

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

Objectif

Développer et tester des modèles pour prédire et expliquer le comportement humain. Le cas d'application est la transition des menus vers les raccourcis claviers.

Compétences acquises à la fin du TP :

- Utiliser des modèles d'apprentissage par renforcement classique en prise de décision et neurosciences
- Développer des modèles pour la prise de décisions séquentielles
- Ajuster des paramètres d'un modèle
- Simuler un modèle
- Comparer des modèles avec les outils log-likelihood et BIC score

Information utile

- Pour chaque question, vous avez une estimation du temps pour la réaliser
- Pour une étape du TP i, vous trouverez un ou plusieurs commentaires dans le code correspondant de la forme # TODO Etape i

À faire avant le TP

- Lire les pages 1, 2, 3 (partie théorique)
- Lire les encadrés théoriques (définition des modèles, définition des métriques)
- Faire l'étape 1 (5mn) pour vérifier que vous avez les bonnes librairies.

Contexte IHM

Raccourcis claviers

Comment encourager les utilisateurs à utiliser les méthodes efficaces à leur disposition pour accomplir une tâche ?
Typiquement, les raccourcis claviers, e.g. *Ctrl+S* pour la commande *Sauvegarder*, sont particulièrement efficaces. Malheureusement, ils sont peu utilisés : beaucoup d'utilisateurs, même experts, continuent à utiliser les menus plutôt que de faire la transition vers raccourcis claviers.

De nombreuses techniques d'interaction sont régulièrement proposées pour favoriser l'utilisation des raccourcis claviers [Bailly et al. 2016]. Par exemple, une technique d'interaction joue (retour audio) le nom du raccourci clavier lorsque la commande est sélectionnée dans le menu (Figure 1).

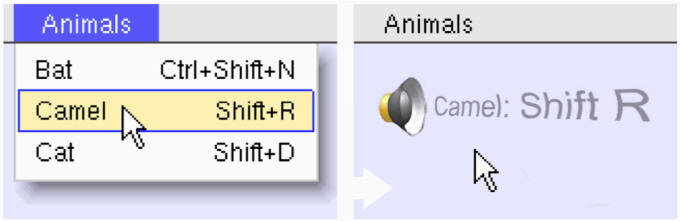

Figure 1: Technique d'interaction jouant le raccourci clavier lorsqu'une commande est exécutée à partir du menu [Grossman et al. 2007]

Les données

Dans [Grossman et al. 2007], les auteurs ont réalisé une étude expérimentale pour comparer plusieurs techniques d'interaction pour favoriser l'usage des raccourcis claviers (Figure 2). Les auteurs comparent trois techniques (Traditional, Audio, Disable). Dans ce TP, nous considérons que les techniques **Traditional** (simple présentation du raccourci à droite de l'item) et **Audio** (Figure 1). Chaque participant teste une seule technique et doit exécuter environ 720 commandes (chaque commande a des fréquences différentes. Des commandes apparaissent fréquemment, d'autres rarement).

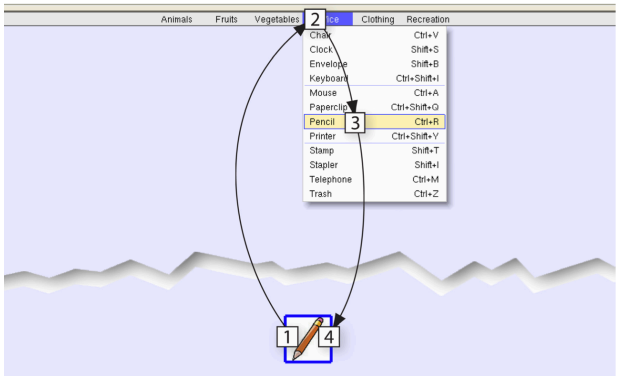

Figure 2: Un essai : Une image est présentée en bas de l'écran. Le participant doit exécuter la commande correspondante à l'aide du menu ou, s'il le connait, du raccourci clavier correspondant.

A chaque essai, le nom d'une commande apparait à l'écran et l'utilisateur l'exécute soit avec la méthode *MENU*, soit avec la méthode *HOTKEY* (raccourci clavier). Mais, d'un point de vue cognitif, on considère que l'utilisateur choisit une de ces trois **stratégies** :

- MENU : je sélectionne la commande avec la méthode MENU.

- HOTKEY : Je sélectionne la commande avec la méthode Raccourcis claviers.
- LEARNING : J'apprends le raccourci clavier en exécutant la commande dans le menu.

On enregistre également pour chaque exécution de commandes, le temps et si la commande a été correctement ou pas. Pour plus d'informations :

- 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

Approche : Apprentissage par Renforcement

La figure 3 illustre le principe de l'apprentissage par renforcement. Étant donné un état (**State**) s, l'agent exécute une action (**Action**) a, reçoit une récompense (**Reward**) r et se retrouve dans un nouvel état s' = s_{t+1}.

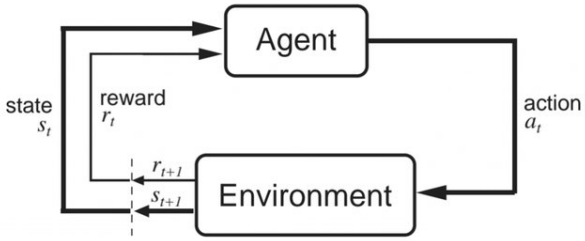

Figure 3: Apprentissage par renforcement

Nous formulons le problème de la transition des menus vers les raccourcis clavier comme un problème d'apprentissage par renforcement où l'interfaces est l'environnement et l'utilisateur est l'agent. Étant donné une commande s à exécuter (State), l'agent choisit une stratégie (Action) parmi MENU, HOTKEY, LEARNING. La récompense (Reward), ici coût, est le temps que met l'agent pour exécuter la commande (pour simplifier nous faisons l'hypothèse que l'utilisateur ne fait pas d'erreur).

Déroulement

Ce TP est en trois parties principales :

1. Étant donné un modèle, ses paramètres et les données des participants, estimer la vraisemblance du modèle, i.e. à quel point le modèle fit les données du participant.
2. Optimiser les modèles, c'est-à-dire trouver les paramètres optimaux du modèle qui maximise la fonction de vraisemblance.
3. Cependant, la vraisemblance n'est souvent pas suffisante pour garantir que ces modèles reflètent bien le comportement de l'utilisateur. Il est nécessaire de les simuler « en roue libre » et de vérifier que les données produites sont compatibles avec données que l'on pourrait observer chez un humain.

Deux modèles sont donnés pour vous familiariser avec les concepts théoriques. Vous implémenterez deux autres modèles. En fonction du temps qui restera, plusieurs pistes vous serons données pour créer vos propres modèles.

TP

1. Se familiariser avec les données (5mn)

1.a Télécharger l'archive <xxx>.

Exécuter le fichier qui charge les données des utilisateurs et les affichent.

> python ./main.py

On voit pour chaque utilisateur, tous les essais réalisés avec le temps de sélection (axe y) et la stratégie utilisée (couleur). Le titre de la figure indique aussi la technique utilisée.

Q0 : Quel est le rôle de la technique d'interaction sur les stratégies utilisées ?

Q1 : Quel est le problème avec cette visualisation ?

1.b Visualisation par commande

Dé-commenter les lignes correspondant à cette question pour résoudre le problème précédent. Cette nouvelle visualisation permet maintenant de voir ce qui se passe pour *chaque commande*. Elle permet d'identifier les commandes fréquentes et les commandes moins fréquentes. Nous utiliserons ces visualisations pour debugger les modèles si besoin.

Q2 : Est-ce que les utilisateurs utilisent davantage les raccourcis claviers pour les commandes fréquentes ?

Nous souhaitons modéliser ces séquences de stratégies pour chaque utilisateur et chaque commande. Nous commencerons par deux moèles : Radom et Choice Kernel.

2. Se familiariser avec le modèle « Random » et le modèle « Choice Kernel » (5mn)

Modèle aléatoire. Ce modèle choisit une stratégie aléatoire avec un biais pour une des options. Dans notre cas, le biais est capturé avec le paramètre HOTKEY_PROB qui est entre 0 et 1 tel que la probabilité d'utiliser la stratégie a pour la commande c :

P(c, a) = HOTKEY_PROB si a == HOTKEY

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

Modèle Choice Kernel (CK). Ce modèle, souvent utilisé en neurosciences, capture la tendance de l'utilisateur à répéter les actions précédentes. L'utilisateur calcule un « choice kernel » $CK_t^k(c)$ qui garde en mémoire la fréquence de l'utilisation de la stratégie (ou action) *k* pour la commande *c* :

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

Où $a_t^k = 1$ si la stratégie k est utilisé à l'essai t, sinon $a_t^k = 0$ et α_{ck} est le taux d'apprentissage (Learning rate). Pour simplifier, on considère que la valeur initiale du choice kernel est 0. On peut alors calculer la probabilité d'exécuter la stratégie a pour la commande c en utilisant la fonction softmax:

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

Où β_{ck} est la température inverse qui le niveau de stochasticité. $\beta_{ck} = 0$ indique que l'option choisie est complétement aléatoire. $\beta_{ck} = \infty$ indique que l'option choisie est celle avec la plus grande valeur. Ce modèle a deux paramètres (α_{ck}, β_{ck}).

Commenter le code correspondant à 1.a et 1.b pour gagner du temps

2.a regarder les fichiers correspondant dans le répertoire /plugin/model/.

Pour chaque modèle, les principales méthodes à réimplémenter sont :

- **action_probs(self, cmd)** qui étant donné une commande à exécuter (un état dans le formalisme RL) retourne la probabilité que l'agent choisisse chaque stratégie (action dans le formalisme RL).
- **time(..)** and **success(..)** pour le reward (vous pouvez ignorer)
- **Update_model()** qui met à jour les variables latentes (e.g $CK_t^k(c)$).

Les paramètres des modèles sont dans /parameters/. La colonne freedom indique si le paramètre est « libre », c'est-à-dire si on peut le faire varier (=1) ou s'il est fixé (=0). Vous pouvez voir qu'il y a plusieurs paramètres fixes pour l'estimation du temps car ce n'est pas le focus de ce TP.

Pour cet exercice, les modèles random et CK sont déjà implémentés. Regarder le code

3. Estimer la vraisemblance des modèles (~25mn)

Étant donné un modèle et ses paramètres, à quel point le modèle fit les données du participant ?

La **fonction de vraisemblance** reflète la capacité du modèle à répliquer les choix d'actions d'un participant d'un essai à l'autre. D'un point de vue bayésien, c'est la probabilité maximale que le modèle *m* choisisse la même série d'actions que le participant. Cela consiste à analyser la prédiction du modèle conditionnellement à un historique. C'est-à-dire estimer la vraisemblance que le participant *p* choisisse l'action a_t^k à l'essai *t* étant donné l'historique ($d_{1:t-1}$) du participant (non du modèle) de l'essai *1* à l'essai *t-1*. Formellement,

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

Où *m* est le modèle, *p* est le participant, θ_m^p est les paramètres du modèle *m* pour le participant *p* et $S_{1:t-1}^p$ est la séquence des actions réalisées par le participant *p*.

Cependant, une limite de cette fonction est qu'elle favorise les modèles « complexes », c'est-à-dire les modèles avec beaucoup de paramètres. Pour cela, on utilise souvent le **Bayesian Information Criterion** (*BIC*) score pour pénaliser les modèles avec de nombreux paramètres :

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

n le nombre d'observations à prédire, k le nombre de paramètres du modèle et *ll(m,p)* la vraisemblance du modèle.

On souhaite calculer la vraisemblance pour chaque participant et chaque modèle.
Attention ! pour analyser les modèles plus précisément, on conservera pour chaque essai la probabilité de choisir l'action du participant.

3.a Implémenter la méthode **run_debug**() de la classe **Individual_Model_Fitting** dans le fichier ./lib/model_fit.py (10-15mn)

Cette méthode prend en entrée :

- La séquence de commandes à exécuter pour un participant (command_sequence)
- Un objet de type User_Output (voir fichier util.py) qui contient trois listes : time, succès et action, décrivant comment le participant a exécuté chaque commande.

La sortie de la méthode est un objet de type **Fit_Output_Debug** qui contient principalement une liste (probs) contenant la probabilité pour chaque essai que le modèle choisisse l'action de l'utilisateur. Ignorer pour le moment l'attribut output.

L'objectif est de mettre à jour la structure Fit_output_Debug afin de contenir la probabilité que le modèle retourne l'action effectuée par le participant. Pour cela, on parcourt la séquence de commande à exécuter et calcule la probabilité que le modèle choisisse l'action de l'utilisateur. On n'oubliera pas de mettre à jour le modèle à l'aide de la méthode **model_update**() avec ce qu'a véritablement fait le participant.

3.b implémenter les méthodes **log_likelihood()** et **bic_score()** dans util.py (2mn)

Quand nécessaire, utilisez des functions de numpy.

3.c Afficher les résultats (5mn)

- Estimer la vraisemblance du modèle à l'aide du code donné dans main.py
- Afficher les résultats à l'aide du code donné dans main.py
- Sauvegarder les graphes (bouton « Save ») pour plus tard.
- Comparer les modèles.

  Q3 Quel est le meilleur modèle ? pour quels utilisateurs ? pour quelles techniques ? Faut-il que la barre soit haute ou basse ? (bien regarder l'intitulé de l'axe y et le signe de la likelihood)

Cependant les paramètres des modèles ont été choisis arbitrairement. Vous pouvez tester *manuellement* d'autres valeurs de paramètres en éditant les fichiers correspondants dans /parameters/. On peut aussi chercher *automatiquement* les paramètres optimaux (étape 4)

4. Optimiser les paramètres des modèles (15mn)

Nous utilisons une méthode d'optimisation pour explorer l'espace des paramètres du modèle et retourner les paramètres qui maximise la fonction de vraisemblance. Cette fonction de vraisemblance est appelée dans la méthode **to_minimize()**. De nombreuses méthodes d'optimisation existent.

4.a Modifier la méthode **optimize()** dans le fichier model_fit.py (5mn)

Nous utiliserons

- la méthode **differential_evolution()** de scipy.optimize (model_fit.py)
- La méthode à minimiser est **to_minimize()** (model_fit.py)

4.b Lancer l'optimisation (2mn)

Exécuter le code correspondant dans main.py. Attention l'optimisation peut prendre du temps (sur mon ordinateur, environ 1mn). Si c'est le cas, les résultats de l'optimisation sont dans le répertoire /optimal_parameters_copy/

4.c Visualiser la vraisemblance avec les paramètres optimaux (5mn)

Les résultats de l'optimisation ont été enregistré dans /optimal_parameters/ (en cas de problème vous avez une copie dans /optimal_parameters_copy/).

- Commenter dans main.py le code correspondant à l'optimisation (4.b)
- Modifier dans main.py le code pour afficher la vraisemblance (3.b) afin de prendre en compte les paramètres optimaux plutôt que les paramètres par défaut. Pour cela, ajouter cette ligne :

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

Q4 Comment à évoluer la vraisemblance ? de manière absolu et relative ? Quel est maintenant le meilleur modèle ? pour quels utilisateurs ? pour quelles techniques ?

5. Simuler le modèle (20mn)

Nous allons maintenant simuler le modèle et voir si les données produites « librement par le modèle » reflètent les données de l'utilisateur. Ici, on ne regarde pas la probabilité que le modèle choisisse l'action de l'utilisateur, mais la séquence d'actions produites librement par le modèle (Deux simulations avec le même modèle / paramètres peuvent donc produire des séquences différentes).
Pour évaluer la séquence d'actions produites, on utilisera une métrique courante pour l'étude des raccourcis claviers : le pourcentage d'utilisation des raccourcis claviers en fonction de la pratique.

5.a Implémenter la méthode simulate() dans le fichier model_simulation.py (10mn)

Vous pourrez vous inspirer de la méthode de ce que vous avez implémentée dans le fichier model_fit.py. Mais attention, ici, le modèle est en roue libre, c'est-à-dire qu'il n'utilise pas les données du participant. Le modèle se met à jour en fonction des actions qu'il a lui-même généré. Vous utiliserez les méthodes de la class Model_Interface:

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

5.b exécuter et visualiser les simulations (10mn)

Utiliser le code fourni dans main-exercice.py. Analyser les figures illustrant l'évolution des raccourcis claviers au cours du temps.

Q5 Quel est le « meilleur modèle » ?

Q6 Est-ce que les modèles vous semblent bons dans l'absolu ?

6 Implémeter le modèle Rescorla-Wagner (10mn)

On aura noté que les modèles précédents n'utilisaient pas de gains (rewards). C'est maintenant le cas avec le Rescorla-Wagner modèle.

Modèle Rescola-Wagner (RW). Ce modèle apprend la *valeur* $Q_t^k(c)$ associée à chaque stratégie *k* et chaque commande *c* en fonction de l'historique des actions et des gains associés. Le modèle utilise alors ces valeurs pour prendre la décision de quelle action effectuée :

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

Où r_t est le gain (reward) et α_{rw} est le taux d'apprentissage (learning rate) entre 0 et 1. On peut alors calculer la probabilité d'exécuter l'action a^k pour la commande c en utilisant la fonction softmax comme pour le modèle Choice Kernel :

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

Où β_{rw} est la température inverse qui le niveau de stochasticité. Ce modèle a deux paramètres (α_{ck}, β_{ck}).

- Implémeter ce modèle. Un template est donné dans ./plugins/models/
- Estimer les paramètres optimaux pour ce modèle or use the ones in /optimal_parameters_copy/
- Estimer et visualiser les résultats de la fonction de vraisemblance et des simulations

Q6 Qu'apprenons nous ? Pourquoi ?

7 Aller plus loin ? (10mn)

Il est possible de combiner ces modèles. En neurosciences, il est fréquent de combiner Rescorla Wagner et Choice Kernel (RWCK). Vous pouvez implémenter ce modèle.

Modèle Rescola-Wagner + Choice Kernel (RWCK). Ce modèle mixe les deux modèles précédents. On utilisera les équations précédentes pour mettre à jour les variables internes. La probabilité de choisir une action est maintenant définie comme :

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

Où β_{rw} est la température inverse qui le niveau de stochasticité. Ce modèle a **quatre** paramètres (α_{rw}, β_{rw}, α_{ck}, β_{ck}).

7 Aller encore plus loin ?

Il est possible d'implémenter d'autres modèles avec différents mécanismes.

- Apprentissage explicite : A chaque fois que l'utilisateur utilise la stratégie LEARNING pour une commande *c*, il apprend le raccourci clavier et donc augmente la « valeur » $Q_t^{HOTKEY}(c)$ avec un learning rate $\alpha_{explicit}$

- Apprentissage implicite. A chaque sélection dans un menu, l'utilisateur apprend implicitement le raccourci clavier. La valeur $Q_t^{HOTKEY}(c)$ augmenterait avec un learning rate $\alpha_{implicit}$ tel que $\alpha_{implicit} \ll \alpha_{explicit}$
- Un mécanisme d'oubli. A chaque essai, l'utilisateur oublie (un peu) tous les raccourcis en mémoire avec taux d'oubli *decay*.
- Un mécanisme que vous souhaiteriez tester.

Des combinaisons qui fonctionnent bien sont :

- Apprentissage explicite + Decay + Choice Kernel
- Apprentissage explicite + Decay + Apprentissage implicite