ML : Upper Confidence Bound

TP4 Apprentissage automatique

Upper Confidence Bound

Jérôme Buisine
jerome.buisine@univ-littoral.fr

21 novembre 2022

Durée : 3h

L'objectif de ce TP est d'exploiter et comparer différentes stratégies afin de maximiser les gains d'un agent dans un contexte d'apprentissage par renforcement.

1 Ressources

Voici un ensemble de ressources qui peuvent vous être utiles durant ce TP :

— 1. Documentation officielle de l'outil `Git` ;
— 2. Gitlab : interface web pour la gestion de projets versionnés sous `Git` ;
— 3. pyenv : permet la gestion d'environnement Python.
— 4. matplotlib : librairie Python qui permet un affichage rapide de données.
— 6. jupyter : un outil de travail permettant une interaction rapide et visuelle avec une console Python.

2 Configuration de l'environnement

Créez un dossier à la racine de votre projet nommé `tp4-ucb`. Dans ce dossier et depuis votre terminal, lancez les commandes suivantes :

```
# spécifie l'environnement virtuel Python à Jupyter
ipython kernel install --user --name=ml-venv
# lance l'application Jupyter
jupyter-lab
```

3 Mise en place du contexte

Créez un nouveau notebook nommé « ucb_bandit.ipynb ». Dans ce notebook l'ensemble des développements seront réalisés dans le cadre du TP.

Pour cette première partie de TP, nous allons utiliser modéliser le problème du bandit manchot au sein d'un casino.

Tâche 1 : Créer une représentation d'une machine bandit dans une classe nommée `Bandit` :

→ Qui prend en paramètre la moyenne et l'écart-type d'une distribution gaussienne.

→ Définir la méthode `draw` qui permet de tirer le bras du bandit et de retourner un gain potentiel. Ce gain sera directement fournit par la distribution et ses paramètres.

Tâche 2 : Créer une représentation d'un casino, à partir d'une classe `Casino` :

→ Un casino est composée de plusieurs machines bandits.

→ Définir la méthode `play` qui permet de tirer le bras d'un bandit ciblé par son index et retourne son gain si le bandit est bien disponible.

→ Créer une méthode de type « `property` », qui permet de retourner le nombre de bandits présents dans le casino.

4 Développements des agents et de leurs strategies

Nous allons maintenant tester un ensemble d'approche pour essayer de maximiser les gains d'un agent présent dans un casino.

Tâche 3 : Créer une classe abstraite `Agent` telle que :

→ Elle est associée à une instance de casino.

→ Enregistre les gains obtenus pour chaque machine bandit présent dans le casino.

→ Permet de connaître le nombre de tour de jeu effectué.

→ Permet l'accès à une propriété `rewards`, qui fournit la somme des récompenses de l'agent.

→ Une méthode abstraite privée `_policy` qui détermine le bandit sélectionné pour un tour de jeu (politique de choix). Elle retourne l'index de ce bandit.

→ Une méthode `select` générique qui permet d'activer la politique de choix du bandit, de jouer ce bandit et de mettre à jour les récompenses reçues pour ce bandit.

→ Une méthode de surcharge `__str__` qui permet un affichage simple de la classe : type actuel de la classe, informations sur les bandits et gain total (vous pouvez revenir sur son développement par la suite).

Voici un exemple d'affichage possible d'un agent confronté à 4 bandits présents dans un casino :

```
Agent007:
- 0: 0.642
- 1: -3.647
- 2: -1962.783
- 3: 2.557
Total: -1960.378 in 1000 rounds.
```

4.1 Agents basés sur des stratégies gloutonnes

Tâche 4 : À partir de cette représentation abstraite de votre agent, créer l'agent `Greedy` (Glouton), qui aura comme politique de choix d'action :

→ Tester une fois chaque machine présente dans le casino.

→ Puis, de toujours sélectionner celle qui lui avait fournie le meilleur gain.

Tâche 5 : Créer maintenant l'agent `EpsilonGreedy`, qui aura comme politique de choix d'action :

→ Tester une fois chaque machine présente dans le casino.

→ Puis, de toujours sélectionner celle qui lui avait fournie le meilleur gain sauf sous contrainte d'un critère probabiliste $\epsilon \in [0, 1]$, de choisir une action aléatoire.

Tâche 6 : Définir un contexte de casino afin de simuler vos stratégies d'agents :

→ Définir 4 bandits ayant respectivement les distributions suivantes : $\mathcal{N}(2, 3)$, $\mathcal{N}(-1, 6)$, $\mathcal{N}(-2, 3)$, $\mathcal{N}(5, 3)$.

→ Simuler pour chaque action les gains potentiels obtenus par vos agents sur 1000 tours (nombre de sélections d'action).

→ Analysez les résultats obtenues en faisant varier le paramètre ϵ de votre agent de type `EpsilonGreedy`.

4.2 Agents avec stratégie adaptative

Nous allons maintenant implémenter un agent ayant une stratégie plus adaptative, appelée UCB1. Pour rappel, dans cet algorithme la formule qui estime la valeur courante d'une action est la suivante :

$$UCB = R_i + c \times \sqrt{\frac{\log(N)}{n_i}}$$

Tâche 7 : Développer l'agent `UCB` afin de le comparer comme précédemment, tel que :

→ Comme pour les autres stratégies, chaque machine est explorée une fois.

→ Puis, la stratégie de choix des actions (politique) est maintenant basée sur la formule UCB.

→ L'action finalement sélectionnée est celle fournissant la valeur maximale UCB.

5 Questions

- Quels sont les défauts des strategies dites gloutonnes ?
- Quel est l'avantage de l'algorithme UCB ?
- Quel est l'impact du paramètre c dans la formule d'estimation de valeur d'une action ?

6 Remise des travaux

N'oubliez pas de réaliser un commit de vos travaux. Taguez également votre projet avec le tag « `tp4` » et soumettez-le sur le serveur Gitlab. Il fera office de rendu.