Manuel de développeur

Robin Courault & Julien Floyd

Sommaire

1. Choix d'implémentation
    1. Architecture
        1. L'Agent
        2. Les Stratégies
        3. Les Environnements
        4. Les Utilitaires
    2. Choix d'utilisation et de fonctionnement
2. Utilisation de la bibliothèque
    1. Utilisation Générale
    2. Utilisation des Environnements
    3. Utilisation des Agents
    4. Utilisation des Stratégies

Choix d'implémentation

Architecture

Pour l'architecture de notre bibliothèque, nous avons fait le choix de conserver le lancement à travers les `Runner`, la classe `MyProp` et la classe principale `App`.

Pour ce qui est de l'architecture des autres éléments, à savoir ceux que nous avons contruits, nous avons créé 4 packages : - `agents` : contient la classe générique d'agent qui utilise les stratégies. - `envs` : contient tous les environnements implémentés par la bibliothèque ainsi que les interfaces et classes abstraites liées. - `strategies` : contient toutes les stratégies implémentées par la bibliothèque ainsi que les interfaces et classes abstraites liées. - `utils` : contient les éléments utilitaires comme les exceptions additionnelles ou les outils.

L'Agent Pour l'agent, nous avons fait le choix d'avoir un agent unique et générique qui utilise un objet `Strategy` pour s'adapter aux différentes stratégies d'apprentissages existantes.

Les Stratégies Pour les stratégies, nous avons fait le choix d'avoir une interface unique `Strategy` qui est ensuite implémentée dans deux classes

abstraites mères `StratBandit` et `StratMDP` qui correspondent aux deux types de stratégies implémentées par la bibliothèque. Chacune de ces classes abstraites mères possèdent les variables et méthodes communes aux stratégies filles spécifiques à chaque stratégie théorique. Cette séparation nous a permis de rapidement construire les différentes stratégies et offre la possibilité d'en ajouter de nouvelles très simplement.

Ainsi nous avons :

```
(Interface) Strategy
    |
    |- (Abstract Class) StratBandit
    |       |- (Class) StratEGreedy
    |       |       \- (Class) StratUCB
    |       |- (Class) StratBanditGradient
    |
     \- (Abstract Class) StratMDP
            |- (Class) StratIterPolicy
            |       \- (Class) StratIterPolicyWithImprovement
            |
            |- (Class) StratIterValue
            \- (Class) QLearning
```

Les Environnements Pour les environnements, nous avons fait le choix d'avoir une classe abtraite mère (`Env`) qui contient les éléments génériques des environnements, et qui implémente donc toutes les méthodes exceptées celles de récompense. Les classes spécifiques (`EnvSimple`, `EnvYT` et `EnvTicTacToe`) implémentent donc la fonction de récompense ainsi que les contructeurs spécifiques éventuels.

Les Utilitaires Nous avons fait le choix de proposer une `RuntimeException` spécifique nommée `ConvergenceAtteinte` qui permet de notifier qu'une convergence a été atteinte. Nous avons laissé cette exception si les utilisateurs de la bibliothèque veulent l'utiliser mais elle n'est plus utiliser par notre bibliothèque.

Choix d'utilisation et de fonctionnement

- Les environnements spécifiques de la bibliothèque (`EnvSimple`, `EnvYT` et `EnvTicTacToe`), possèdent chacun des constructeurs qui construisent des agents et leurs stratégies en fonction d'un argument `typeAlgo` qui prend des chaînes de caractères prédéfinies et crée la stratégie souhaitée parmi celles de la bibliothèque (voir description des constructeurs dans le code).
- Comme avec le code fourni, il suffit de faire `mvn clean compile exec:java` dans le dossier `Projet/rl-lib/` pour compiler et lancer

l'application `App` qui exécute `SimpleRunner`, `YTRunner` et si nous avons terminé de l'implémenter, également `TicTacToeRunner`.

Utilisation de la bibliothèque

Utilisation Générale

Pour utiliser la bibliothèque, il est nécessaire d'avoir un objet `Env`, contenant lui même un objet `Agent`, qui contient lui même un objet implémentant l'interface `Strategy`.

Pour construire un objet `Env`, il vous faut également un état initial qui nécessite le chargement d'une machine ProbB et son initialisation via les objets `Runner`.

Une fois tous les objets construits, l'apprentissage de l'agent se fait en appelant sa méthode `learn`, si vous ne possédez que l'environnement, vous pouvez utiliser la succession de méthodes suivante : `env.getAgent().learn()`.

Vous pouvez ensuite imprimer les résultats de l'apprentissage en utilisant :

- `env.printAgent()`
- `System.out.println(env.getAgent())` ou `System.out.println(agent)`

Pour utiliser l'agent sur un autre environnement, récupérez l'agent (`env.getAgent()`), puis créer un nouvel environnement avec l'agent. Il vous suffit ensuite d'exécuter : `env.execAction(env.getAgent().choose(env.getActions))`, cela va faire en sorte que l'agent choisisse la prochaine action dans l'état courant de l'environnement puis que ce dernier applique ce choix pour passer au prochain état. Il suffit donc de répéter l'opération pour appliquer l'apprentissage de l'agent sur l'environnement.

Pour utiliser l'agent sur le même environnement, il suffit de faire `env.reset()` puis d'exécuter la suite de méthodes mentionnée précédemment pour demander à l'agent de faire des choix.

Veuillez noter que lorsque vous demandez à l'agent de faire un choix sur un ensemble d'actions en utilisant `choose()`, l'agent ne modifie pas sa table et n'apprend donc pas.

Utilisation des Environnements

Tous les environnements non abstraits de la bibliothèques sont utilisables directement si leurs paramètres correspondent à ce que vous souhaitez, il est également tout à fait possible de les étendre notamment au niveau des constructeurs pour ajouter vos éventuelles nouvelles stratégies.

Si vous souhaitez, utiliser votre propre environnement, étendez la classe abstraite `Env` et implémentez ou surchargez ce dont vous avez besoin. Si vous étendez `Env`, il faut obligatoirement implémenter la fonction de récompense `getReward(Transition)`. Il est également conseillé mais pas obligatoire d'implémenter un constructeur plus avancé que celui de `Env`, car l'agent n'est

pas défini (il est égal à null) par défaut au moment de la création d'un objet `Env`.

Voir le code (Env.java) pour la description des méthodes, en voici cependant la liste :

- `(constructor)` Env(State initialState)
- void setAgent(Agent agent)
- void execAction(Transition transToApply)
- State simulActionOnCurrent(Transition transToApply)
- State simulAction(State state, Transition transToApply)
- double getReward(Transition transition)
- List getActions()
- List getActions(State state)
- void printAgent()
- State getCurrentState()
- void reset()
- Agent getAgent()

Utilisation des Agents

L'agent de la bibliothèque est utilisable directement en le construisant à partir d'un environnement (`Env`) et d'une stratégie (`Strategy`). Vous pouvez étendre cet agent si vous souhaitez le surcharger ou lui ajouter des fonctionnalités.

Voir le code (Agent.java) pour la description des différentes méthodes, voici cependant leur liste :

- `(constructor)` Agent(Env env, Strategy strategy)
- `(constructor)` Agent(Env env, Strategy strategy, int maxIterations)
- void learn()
- Transition choose(List)
- String toString()

Utilisation des Stratégies

Toutes les stratégies non abstraites de la bibliothèques sont utilisables directement si leurs paramètres correspondent à ce que vous souhaitez, il est également tout à fait possible de les étendre pour les surcharger ou les agrémenter.

Les stratégies abstraites `StratBandit` et `StratMDP` sont communes à plusieurs stratégies de la bibliothèque et vous pouvez également les étendre pour utiliser les fonctionnalités et variables qu'elles intègrent déjà afin de plus rapidement constituer des stratégies fonctionnelles.

Enfin, vous pouvez construire votre propre stratégie depuis presque zéro en implémentant l'interface `Strategy` dont voici la liste des fonctions (pour plus de précision et une petite description, allez voir le code (Strategy.java)) :

- void learn()

- Transition choose(List actions)
- boolean convergenceAtteinte()
- String printTable()