TP 3 : Implementation de la méthode de Monte Carlo

(a rendre à l'adresse : santucci_j@univ-corse.fr)

But du TP : apprendre à jouer au blackjack avec la méthode de prédiction de MC

0. Règles du jeu de Blackjack

Le jeu se compose d'un joueur (player) et d'un croupier (dealer) .
Le but du joueur est que la valeur de la somme de toutes ses carte soit égale à 21 ou une valeur supérieure à la somme des cartes du croupier sans dépasser 21.
Si l'un des ces critères est rempli, le joueur gagne la partie ; sinon le croupier gagne (si on dépasse 21 on perd et on appelle cela faire faillite).
.

Voyons plus en détail.
La valeur des cartes valet (J comme Jack) , dame(Q comme Queen) et roi (K comme King) sont considérées comme 10.
La valeur de l'AS(A) peut être 1 ou 11 selon le choix du joueur.
Donc le joueur peut décider si la valeur 'un As doit être 1 ou 11 pendant la partie.
La valeur du reste des car§es (de 2 à 10) est juste la valeur nominale (la valeur de la carte 2 est 2, la valeur de la carte 3 est 3,, etC…).

Le jeu se compose d'un joueur et d'un croupier : il peut y avoir plusieurs joueur mais un seul croupier.

Tous les joueurs s'affrontent avec le croupier et non avec d'autres joueurs.

Considérons le cas d'un seul joueur et du croupier.

Examinons les différents cas (nous sommes le joueur et nous sommes donc en concurrence avec le croupier).

CAS N°1 : le joueur gagne
Un joueur reçoit 2 cartes au départ.
Ces deux cartes sont face visibles c-a-d quelles deux cartes du joueur sont visibles pour le croupier.
De même le croupier reçoit aussi 2 cartes. MAIS l'une des cartes du croupier est face visible et l'autre est face cachée. C'est à dire que le croupier ne montre qu'une seule de ses cartes.
Par exemple dans la figure suivante, le joueur a deux cartes (toutes deux face visible) et le croupier a également deux cartes (une seule face visible):

Les actions à effectuer pour jouer soit pour le joueur sont **Hit** ou **Stand**.
L'action **Hit** consiste à obtenir une carte de plus.

L'action **Stand** consiste à considérer que nous ne voulons plus d'autre carte ce qui va impliquer que le croupier montre toutes ses cartes. Celui qui a une somme de cartes d'une valeur de 21 ou supérieure à celle de l'autre mais n'excédant pas 21 remporte le jeu.

Nous avons vu que
la valeur de J, K et Q est 10. Comme on le voit sur la figure précédente, nous avons les cartes J et K dont la somme est 20. Donc la valeur totale de nos cartes est déjà un grand nombre et elle n'a pas dépassée 21. Donc on choisit l'action **Stand** ; cette action indique au croupier qu'il doit montrer ses cartes.

Comme on le voit dans la figure ci-dessous le croupier a montré ses cartes et la valeur totale est de 12 (< 20) . DONC le joueur gagne la partie.

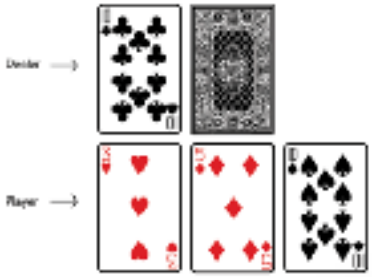
CAS N°2 : le joueur perd

La figure ci-dessous décrit une nouvelle donne :

Nous devons décider quelle action à effectuer (Hit ou Stand). Dans la figure ci-dessus nous nous voyons que nous avons deux cartes K et 3 qui donc totalisent 13. En étant optimiste on peut espérer que la valeur totale des cartes du croupier ne sera pas supérieure à 13. Alors on choisit l'action Stand. et donc le croupier doit montrer ses cartes comme on peut le visualiser à la figure ci-dessous. Or totale du croupier 17 qui est supérieure à 13. Donc le joueur perd.

CAS N°3 : le joueur fait faillite
La figure ci-dessous décrit une nouvelle donne :

Nous devons décider quelle action à effectuer : Hit ou Stand.
On remarque que la valeur de nos cartes est de 8 et celles du croupier sont au moins de 10. Donc on choisit l'action Hit pour augmenter notre leur totale (sachant qu'on ne peut pas perdre (faire faillite) puisque on ne peut pas dépasser 21. Le résultat de l'action Hit est donné ci-dessous.

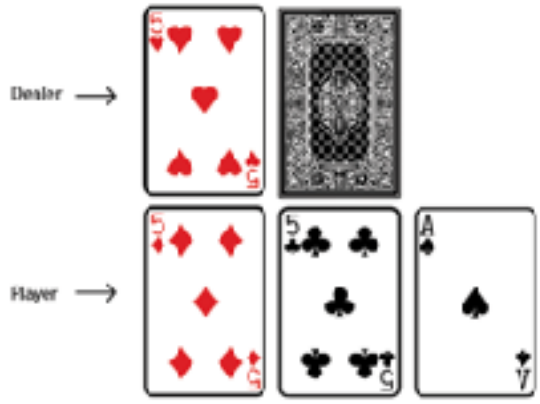


Maintenant la valeur totale de nos cartes est de 18. Nous devons encore une fois décider l'action à effectuer (Hit ou Stand). Si le croupier tire une carte de valeur 9,10 ou 11 il a gagné. Alors décidons d'effectuer encore une fois l'action Hit. Afin que nous puissions obtenir une valeur totale de nos cartes un peu plus grande (avec le risque de faire faillite - dépasser 21).
Nous obtenons la configuration suivante:

L'action Hit nous a donnée une carte supplémentaire mais maintenant la valeur totale de nos cartes est égale à 28 et donc dépasse 21 : c'est ce qu'il s'appelle faire faillite et donc on a perdu la partie.

CAS N°4 : L'as utilisable.
Nous avons vu que la valeur e l'As pouvait être 1 ou 11 et que le joueur (ou le croupier) peut décider de la valeur de l'As pendant la partie.
Voyons comment cela fonctionne. Supposons la donne suivante :

La valeur totale de nos cartes est de 10. Donc on choisit l'action Hit pour augmenter cette valeur (de plus il est impossible que nous faisions faillite).
La figure suivante nous montre ce que nous obtenons pares avoir reçu une nouvelle carte ;

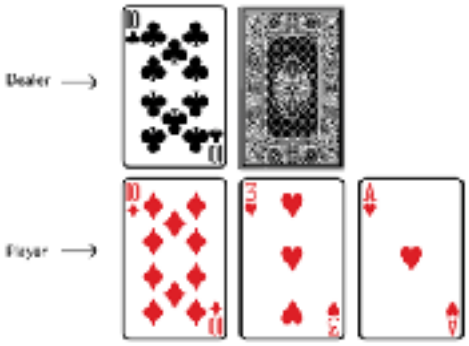


La nouvelle carte est un As. Nous devons décider quelle sera la valeur de cet As : 1 ou 11. Si on choisit 1, la valeur totale des cartes sera 11 mais si on choisit 11 alors la valeur totale est 21. Donc on choisit 11 et on gagne la partie.
Cet As est appelé l'As utilisable car il nous a aidés à gagné la partie.

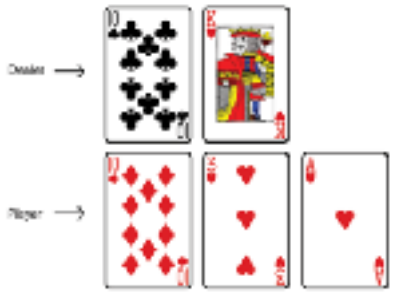
CAS N°5 : L'as non utilisable.

Considérons la nouvelle donne suivante:

La somme des valeurs de nos carte est de 13. On choisit l'action Hit pour essayer de l'augmenter. On obtient alors la configuration suivante après avoir reçu la nouvelle carte qui est un As



Nous devons maintenant décider de la valeur de l'As : 1 ou 11. Si on choisit 11, la somme des la valeurs des cartes est de 23 et donc dépasserait 21 et donc nous perdrions. Donc au lieu de choisir 11, nous choisissons 1 et la somme est alors de 14. Nous devons encore une fois décider quelle action effectuer (Hit ou Stand).
Disons que nous espérons que la valeur de la somme du croupier sera inférieure à 14.
Donc on choisit Stand et le croupier doit montrer ses cartes comme décrit ci-dessous:



La somme des cartes du croupier est de 20 contre 14 pour nous. Donc on perd la partie et dans ce cas l'As est appelé As non utilisable car il ne nous a pas aider à gagner la partie.

CAS N°6 : Match nul
Si la valeur de la somme des cartes du joueur et du croupier est la même, alors il y a match nul.

1. L'environnement Blackjack dans Gym

O£n va utiliser un environnement de la Library Gym qui correspond au jeu Blackjack.
Lid du jeu est Blackjack-v0 pour python 2.7 ou Blackjack-v1 pour les versions de python3.
Par exemple si vous entrez la séquence suivante:

```
>>> import gym
>>> env = gym.make('Blackjack-v1')
>>> print (env.reset())
((8, 3, False), {})
>>>
```

Vous remarquez que vous obtenez pour l'état quelque chose de la forme : ((8,3,False), {}).
Ce format décrit l'état du jeu : ((somme des cartes du joueur, valeur de la carte visible du croupier, Booleen), informations éventuelles).
Le booléen est True si le joueur à un as utilisable (valeur 11) et False sinon.

Pour ce qui concerne les actions :

```
>>> print (env.action_space)
Discrete(2)
```

Il y a bien 2 action (Stand qui sera l'action 0 et Hit qui sera l'action 1.

En ce qui concerne le reward nous aurons :

- +1 si le joueur gagne
- -1 si le joueur perd
- 0 si il. y match nul.

2. Implémentation de la prediction Every-visit sur le blackjack

Vous pouvez avoir besoin des bibliothèques suivantes pour implementer la prédiction Every-visit.

```
>>> import pandas as pd
>>> from collections import defaultdict
>>>
```

2.1. Définition d'une politique

Dans la méthode de prédiction, nous avons vu en cours qu'en entrée nous avons une politique et nous allons prédire la value fonction de la politique donnée en entrée.

Donc vous allez tout d'abord définir un fonction (policy function) qui nous servira de politique d'entrée.
Donc vous allez définir la politique d'entrée dont la value fonction sera prédite.

La fonction policy que vous allez définir prend en entrée un état (donc de la forme ((8,3,False), {}))
La fonction retourne l'action 0 (Stand) si la somme des cartes du joueur est supérieure à 19 et retourne action 1 (Hit) sinon.

Nous avons défini une politique cohérente : il est plus judicieux d'effectuer une action 0 (Stand) lorsque notre somme est déjà supérieure à 19 car en recevant une nouvelle carte, nous avons de grande chance de perdre la partie.

2.2. Générer un épisode

Vous allez maintenant générer un épisode en utilisant la politique donnée. Donc vous devez écrire la fonction generate_episode (policy).

Il vous faudra fixer le nombre de pas de temps (par exemple num_timesteps = 100).

Pour définir l'épisode vous allez choisir une structure de donnée pour stocker l'épisode : par exemple episode = []

Il faudra initialiser l'état (state = env.reset()) puis entrer dans la boucle for.
Vous allez pouvoir sélectionner une action (action) à partir de la politique d'entrée , exécuter l'action (avec env.step(action)) et récupérer les information concernant l'etat suivant obtenues par env.step(action) : donc le nouvel état, le reward, done, info, commentaires donc next_state, reward, done, info, com.

nous avons besoin de l'état, d l'action et du reward correspondant pour définir l'épisode donc de state, action et reward.

Si on a trouver l'état final (done = True on sort de la boucle sinon on recommence avec le nouvel état (state = next_state.

Donc votre fonction generate_episode (policy) doit retourner quelque chose d cela forme [(10,2,False), 1,0), ((20,2,False),0,1.0)]
Cela voudra dire qu'on deux états dans l'épisode généré. On exécute l'action 1 dans l'état (10,2,False) et on reçoit 0 comme reward, et nous exécutons l'action 0 dans l'état (20,2,Flase) et on reçoit 1 comme reward.

2.3. Calcul de la Value fonction.

Nous avons vu en cours que pour prédire la value fonction, il faut générer plusieurs épisodes à l'aide de la politique donnée en entrée et calculer la state value en tant que rendement moyen sur plusieurs épisodes.

Afin de respecter l'algorithme vu en cours, vous allez devoir définir définir : Total_return et N (et donc choisir une structure de données - (par exemple un dictionnaire puisque total_return et N sont associées à un état donné).

il vous faut aussi définir le nombre d'itérations qui correspond au nombre d'épisodes générés (par exemple num_iterations = 500000)

Comme dans l'algorithme du cours, vous allez faire une boucle for sur le nombre d'iterations.
Il faut pour chaque itérations générer ion épisode avec generate_episode(policy) puisque la politique (policy) a été défini auparavant.
Vous allez stocker tous les états, actions et rewards obtenus à partir de cet épisode
Puis pour chaque pas de l'épisode (dont on connait l'etat courant st) , vous allez calculer le return R de l'état courant st comme indiqué dans l'algorithme (R(st) = sum(rewards[t:].

Vous allez mettre à jour le total_return de l'état courant (dans l'algorithme total_return(st) = total_return(st) + R(st).
On doit aussi mettre à jour le nombre de fois que l'état courant est visité : N(st) = N(st) + 1.

On peut alors calculer V(s) = total_return(s) / N(s).

Donc vous avez maintenant la state value des états qui correspond à la moyenne du return de l'état a travers plusieurs épisodes.

On a donc prédit la value fonction d'une politique donnée en utilisant la méthode MC every-visit

2.4. Verification sur la Blackjack

Vous allez vérifier votre implémentation sur l'exemple du Blackjack

3. Implémentation de la prediction Every-visit sur le blackjack

3.1. Prédiction de la value fonction en utilisant lamérhode MC first visit.

Vous allez maintenant écrire la fonction qui permet de prédire la value fonction en utilisant la méthode MC first visite.
L'algorithme est exactement le même comme vu en cours sauf que nous calculons le retour d'un état uniquement pour sa première occurence dans l'épisode.

Il vous suffira de modifier le code précédent afin de calculer le return seulement pour la première occurence.

3.2. Vérification sur le Blackjack

Vous allez vérifier votre implémentation sur l'exemple du Blackjack

4. Implémentation de l'algorithme MC de commande avec une politique epsilon-greedy on policy (vue en cours)

4.1. Librairies

Vous aurez bien sur besoin d'importer les librairies nécessaires. Par exemple:

```
>>> import gym
>>> import pandas as pd
>>> from collections import defaultdict
>>> env = gym.make('Blackjack-v1')
```

4.2. Définition d'une fonction epsilon_greedy_policy

Nous avons vu en cours que nous devions sélectionner les actions à partir d'une politique epsilon-greedy. Donc vous devez écrire une fonction appelée epsilon_greedy_policy qui prend comme arguments l'état et la Q value (table des valeurs) et qui retourne l'action à exécuter dans l'état passé en parametre.

Vous pouvez vous inspirez du code de la fonction donnée en cours.

4.3. Génération d'un épisode

Vous allez maintenant écrire la fonction qui permet de générer un épisode en utilisant la politique epsilon_greedy.

Appelons generate_episode la fonction qui prend la Q value (table des valeurs) en entrée et renvoie l'épisode.

Il vous faudra une variable num_timesteps qui indique le nombre d'etapes maximum de l'épisode (par exemple num_timesteps = 100)

Vous pouvez vous inspirez de la fonction de génération d'épisodes précédente (voir 2.2) sauf que la fonction prend la table des Q value en paramètre.
donc il faudra définir generate_episode(Q).

Vous utiliserez la fonction de 4.2 epsilon_greedy_policy (state,Q) pour selection l'action .

4.4 . Calcul de la politique optimale.

Nous avons vu en cours que dans la méthode MC de commande On-policy , aucune politique n'est donnée en entrée.
Donc il faut initialiser une politique aléatoirement lors de la première itération et améliorer, la politique de manière itérative en calculant la table Q value.

Puisque nous allons extraire la politique à partir de la Q fonction, nous n'avons pas à définir la politique de manière explicite.
A mesure que la Q value s'améliore, la politique s'améliore également implicitement.

Donc dans la première itération vous allez générer l'épisode en extrayant la politique (epsilon-greedy) de la table Q initiale.

Apres une série d'itérations, vous allez trouver la Q fonction optimale et donc la politique optimale.

Donc on va passer en paraître la Q table à la fonction generate_episode.
donc par exemple épisode = generate_episode(Q).

Vous allez ensuite récupérer toutes les paires état-action de l'épisode (facile)
De la même façon, il vous faudra aussi stoker tous les rewards obtenus dans l'épisode dans la liste des rewards.

Puis pour chaque pas de l'épisode, vous allez calculer le return R de l'état courant st comme indiqué dans l'algorithme (R(st,at) = sum(rewards[t:].

Vous allez mettre à jour le total_return de la paire état-action (dans l'algorithme du cours : total_return(st,at)= total_return(st,at) + R(st,at).
On doit aussi mettre à jour le nombre de fois que l'état courant est visité : N(st,at) = N(st,at) + 1.

On peut alors calculer la Q value = total_return(st,at) / N(st,at).

Donc à chaque itération , la Q value s'améliore et donc la politique aussi.

4.5. Vérification sur le Blackjack

Vous allez vérifier votre implémentation sur l'exemple du Blackjack