teaching-iaro (/github/nmaudet/teaching-iaro/tree/master)
/
3-dynamiquesMultiagents (/github/nmaudet/teaching-iaro/tree/master/3-dynamiquesMultiagents)

In [1…
```
%load_ext notexbook
%texify
```

Dynamiques multiagents

1. Meilleures réponses avec plusieurs agents

On a vu dans les cours précédents la notion de meilleure réponse. Remarquons tout d'abord que cette notion s'étend tout à fait naturellement à plus de deux agents.

Lorsque le système multiagent comporte plus de deux agents, on peut construire de manière similaire les tables de meilleures réponses, mais selon les possibilités de stratégies de **tous** les autres agents. Par exemple, pour décrire les meilleures réponses d'un agent x aux actions de x_1 , x_2 , x_3 , on aura une table de ce type:

BR(x)	x_1	x_2	x_3
acheter	emprunter	emprunter	emprunter
emprunter	acheter	acheter	emprunter
…	…	…	…

Problème: la table de meilleure réponse grandit exponentiellement avec le nombre d'agents.

Toutefois, dans de nombreuses situations, les actions des agents n'ont que des conséquences locales. On peut donc limiter la représentation des meilleures réponses aux agents concernés (par exemple aux voisins sur un graphe). On parle alors de **jeux graphiques**.

Exemple. Imaginons par exemple que Riri, Fifi, et Loulou habitent dans une rue: Riri au début de la rue, Fifi au milieu de la rue, et Loulou à la fin de la rue. Les trois amis se demandent s'ils doivent acheter une tondeuse à gazon ou plutôt l'emprunter à leur voisin. Cette décision dépend du choix de leur voisin. Riri est voisin de Fifi, Fifi est voisin de Riri et Loulou, et Loulou n'est voisin que de Riri.

In [2…
```
def bestResponse(s,agent):
    if agent == 'riri' or agent=='loulou':
        if s['fifi']=='achete':
            return 'emprunte'
        else:
            return 'achete'
    elif agent == 'fifi':
        if s['riri']=='emprunte' and s['loulou']=='emprunte':
            return 'achete'
        else:
            return 'emprunte'
```

2. Dynamiques de meilleures réponses

On peut sur cette base étudier le processus dynamique qui résulte du fait que les agents jouent itérativement des meilleures réponses.

De manière générale, on peut parler **partial best-response**: une partie de la population d'agents joue une meilleure réponse à l'état courant. On a donc deux cas limites:

- la situation où **un seul agent** est considéré à chaque fois: les

actions sont donc modifiée

- la situation où **tous les agents** modifient leurs stratégies simultanément à chaque fois, en réponse à l'observation de l'état courant.

Dans la version **agent unique**, l'algorithme de dynamique de meilleure réponse prend donc la simple forme suivante:

In [2…

```
agents = ['loulou', 'riri', 'fifi']
s= {'riri': 'emprunte', 'fifi': 'emprunte', 'loulou': 'emprunte'}
equilibrium = False
print (s)
while not equilibrium:
    equilibrium = True
    for i in agents:
        if s[i]≠bestResponse(s,i):
            s[i]=bestResponse(s,i)
            equilibrium=False
        print (s)
```

```
{'riri': 'emprunte', 'fifi': 'emprunte', 'loulou': 'emprunte'}
{'riri': 'emprunte', 'fifi': 'emprunte', 'loulou': 'achete'}
{'riri': 'achete', 'fifi': 'emprunte', 'loulou': 'achete'}
{'riri': 'achete', 'fifi': 'emprunte', 'loulou': 'achete'}
{'riri': 'achete', 'fifi': 'emprunte', 'loulou': 'achete'}
{'riri': 'achete', 'fifi': 'emprunte', 'loulou': 'achete'}
{'riri': 'achete', 'fifi': 'emprunte', 'loulou': 'achete'}
```

> Quelle est l'autre équilibre de Nash de ce jeu? Pouvez-vous trouver une séquence des agents menant (depuis le même état initial) à cet équilibre? Que se passe-t-il sur cet exemple si on considère la dynamique où tous les agents modifient simultanément leurs stratégies?

Quelques propriétés simples des dynamiques de meilleures réponses:

- Lorsqu'un équilibre est atteint, il s'agit bien d'un équilibre de Nash (par définition).
- Les dynamiques de meilleures réponses cyclent nécessairement lorsqu'il n'y a pas d'équilibre de Nash
- Les dynamiques de meilleures réponses peuvent cycler *même si il existe des équilibres de Nash*
- Toutefois, sous certaines conditions, on peut montrer qu'elles convergent nécessairement vers des équilibres de Nash.

3. Dynamiques de réponses améliorantes

Au lieu de supposer que les agents changent de stratégie en choisissant la meilleure (ou une des meilleures s'il en existe plusieurs) réponse améliorante, on peut supposer que les réponses sont seulement améliorantes (donc meilleures que la stratégie actuelle, sans être nécessairement *la* meilleure)

Dynamiques de mariages stables

Reprenons le cadre des mariages que vous avez étudié en détail dans la première partie de ce cours, et essayons de définir ce que pourrait être une dynamique de meilleure réponse, ou de réponse améliorante.

On rappelle qu'une paire instable est une paire d'agents qui préfèreraient être ensemble qu'avec leur partenaire actuel. Dans notre cadre, on dit qu'on satisfait une paire lorsque l'on associe les deux agents de cette paire. Les anciens partenaires peuvent alors se trouver associés ou non. Vous avez vu en TD la procédure des **divorces successifs**, qui se base précisément sur ce principe (avec remariage des anciens partenaires).

De manière générale, on a donc le cadre suivant:

- on part d'une situation initiale, c'est-à-dire un matching (éventuellement partiel) entre les agents de $\mathbf{M}$ et $\mathbf{W}$.
- on considère une paire d'agents instable $(\mathrm{m}, \mathrm{w}) \in \mathbf{M} \times \mathbf{W}$.
- dans une dynamique de **réponse améliorante**, on satisfait cette paire instable.
- dans une dynamique de **meilleure réponse**, on considère un agent impliqué dans cette paire instable (par exemple m), et on satisfait sa paire instable préférée parmi toutes les paires instables dans lesquelles il est impliqué.

Knuth (1976) a montré que la dynamique de réponse améliorante ne converge pas forcément, et vous l'avez aussi constaté en TD. Son exemple original était le suivant (en débutant du mariage $(\mathrm{m}_1, \mathrm{w}_1), (\mathrm{m}_2, \mathrm{w}_2), (\mathrm{m}_3, \mathrm{w}_3)$):

Women	**Men**
w1: m1, m3, m2	**m1**: w2, w1, w3
w2: m3, m1, m2	**m2**: w1, w3, w2
w3: m1, m3, m2	**m3**: w1, w2, w3

Le même type de cyle peut être obtenu pour la dynamique de meilleure réponse (Ackerman et al., 2008). Voici l'exemple:

Women	**Men**
w1: m2, m3, m1	**m1**: w1, w3, w2
w2: m1, m2, m3	**m2**: w2, w1, w3
w3: m3, m1, m2	**m3**: w1, w2, w3

Un cycle est le suivant (le . représente le fait que le femme d'indice i n'est pas en couple):
$(. , \mathrm{w2} - \mathrm{m2}, \mathrm{w3} - \mathrm{m3}) \rightarrow (\mathrm{w1} - \mathrm{m3}, \mathrm{w2} - \mathrm{m2}, .) \rightarrow (\mathrm{w1} - \mathrm{m3}, \mathrm{w2} - \mathrm{m1}, .)$
$\rightarrow (. , \mathrm{w2} - \mathrm{m2}, \mathrm{w3} - \mathrm{m1}) \rightarrow (. , \mathrm{w2} - \mathrm{m2}, \mathrm{w3} - \mathrm{m3})$

De là découlent de nombreuses autres questions naturelles:

- peut-on toujours une séquence qui mène à un état stable?
- si la dynamique est aléatoire, quelle est la probabilité d'atteindre un état stable?
- combien de temps peut-il falloir pour atteindre un état stable?
- si on part d'un état initial particulier (par exemple le mariage "vide"), est-ce que cela peut changer les choses?

4. Dynamiques avec apprentissage

> Cette partie s'appuie notamment sur le Chapitre 7 de Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations (Shoham and Leyton-Brown)

Das le cadre de telles dynamiques, il est naturel d'envisager que les agents vont pouvoir tirer profit des interactions passées pour décider de l'action à sélectionner. Il s'agit donc d'apprentissage. Reprenons le cadre du **dilemme du prisonnier itéré** que nous avons étudié au cours précédent.

4.1 Caractérisation des stratégies des agents

Reprenons les stratégies envisagées dans le cadre du dilemme du prisonnier. Rappelons que ces stratégies peuvent se baser sur les histoires (les séquences) des coups joués. On voit que l'on peut les classer selon deux critères:

- le **niveau d'information** requis par la règle: on parle d'une règle de niveau d'information k lorsque les k derniers coups de l'adversaire sont utilisés.

- le fait que la règle soit **déterministe** (prescrive une action unique) ou **stochastique** (donne une distribution de probabilité sur des actions possibles à jouer)

On peut parler d'apprentissage dès qu'une règle utilise la connaissance des coups joués par les autres joueurs, autrement dit dès qu'elle est de niveau $k \geq 1$. Lorsque $k = 0$, la stratégie est dite **stationnaire**.

> Pouvez-vous caractériser selon ces critères les règles random, tit-for-tat, toujours trahir...?

4.2 Fictitious Play

Supposons maintenant que les agents apprennent plus spécifiquement le comportement des autres joueurs, et jouent la meilleure réponse à la prédiction qu'ils font du comportement des autres agents. Un des modèles les plus simples d'un tel type d'apprentissage a été proposé par Brown (1951) sous le nom de *fictitious play*.

Il repose donc sur les phases suivantes:

- prédiction des actions des autres joueurs
- choix de la meilleure réponse à la stratégie des autres agents
- observation des actions des autres joueurs, mise à jour de la prédiction

(Standard) Fictitious Play: Nous présentons ici le modèle dans un contexte où seuls deux agents sont en interaction à chaque fois, même si le système comprend plus d'agents. (Le modèle s'étend à des interactions impliquant plus d'agents, mais il faut alors faire des choix sur la manière de combiner les probabilités assignées aux différents agents).

- Il faut d'abord que l'agent fasse une prédiction *a priori* sur l'action initiale de l'agent (cette prédiction peut être une

distribution de proba, pas nécessairement un coup donné).

- Chaque joueur tient à jour la fréquence de sélection des actions et joue sa meilleure réponse à ce modèle probabiliste.

Ici la meilleure réponse est entendu en terme de **gain espéré** pour chacune de ses actions. L'hypothèse est donc que l'autre agent joue une stratégie stationaire.

Exemple:

Plaçons nous du point de vue l'agent R, qui est opposé à l'agent B. La table suivante donne les gains pour le joueur R.

R \ B	**c**	**d**
a	7	10
b	10	0

Supposons que la stratégie de B soit de jouer c et d à 50/50. Et que R ait comme a priori que B va presque certainement toujours jouer c ($P(c) = 0.8$), qui provient d'un échantillon initial de 4 c et 1 d.

#c	**#d**	**P(c)**	**P(d)**	**BR(R)**	**coup B**
4	1	0.8	0.2	b	c
5	1	0.83	0.17	b	d
5	2	0.71	0.29	a	c

Pourquoi a est-elle la meilleure réponse à l'itération 3?

On voit qu'à cette étape, R estime que B joue c et d avec une probabilité 50/50. Quelle est le gain espéré pour lui de jouer

- l'action a: $0.71 \times 7 + 0.29 \times 10 = 7.86$
- l'action b: $0.71 \times 10 + 0.29 \times 0 = 7.1$

On a donc gain espéré pour a > gain espéré pour b. La meilleure réponse est de jouer a.

Et ainsi de suite pour les itérations suivantes...

#c	**#d**	**P(c)**	**P(d)**	**BR(R)**	**coup B**
6	2	0.75	0.25	a	d
6	3	0.66	0.33	a	c
...	...	...	...	...	...

> Mais notons que l'apprentissage reste rudimentaire. Par exemple, si j'observe que l'autre joueur joue *alternativement* c et d, un agent utilisant fictitious play attribuera à la limite une probabilité 50/50 pour l'autre agent de jouer c ou d, alors que s'il avait reconnu la séquence, il serait capable d'obtenir un bien meilleur gain.

In [2…

```
actions = [('a','b'),('a','b')]
matGains ={'a':{'a':(3,2),'b':(0,0)},'b':{'a':(0,0),'b':(2,3)}}
prior = [{'a':3,'b':1},{'a':1,'b':2}]
nbJoueurs = 2
nbObservations = [sum (prior[i].values()) for i in range(nbJoueur
nbIterations = 10

def fictitious(observations):
    actionsChosen =[]
    for i in range(nbJoueurs):
        expectedGain = {'a':0,'b':0}
        bestAction = actions[i][0]
        bestGain =0
        print ("\njoueur ",i, end=" | ")
        for actionMe in actions[i]:
            for actionOther in actions[-i]:
                expectedGain[actionMe] += (observations[i][action
            print ("gain espéré pour ",actionMe, "%0.2f" % expect
            if expectedGain[actionMe]>bestGain:
                bestAction=actionMe
                bestGain=expectedGain[actionMe]
        actionsChosen+= bestAction
    return actionsChosen

observations = prior.copy()
for i in range(nbIterations):
    print (observations)
    nextAction0, nextAction1 = fictitious(observations)
    print ("\n", nextAction0,nextAction1)
    observations[0][nextAction1]+=1
    observations[1][nextAction0]+=1
    nbObservations[0]+=1
    nbObservations[1]+=1
```

```
[{'a': 3, 'b': 1}, {'a': 1, 'b': 2}]

joueur  0 | gain espéré pour  a 2.25 | gain espéré pour  b 0.50 |
joueur  1 | gain espéré pour  a 0.67 | gain espéré pour  b 2.00 |
 a b
[{'a': 3, 'b': 2}, {'a': 2, 'b': 2}]

joueur  0 | gain espéré pour  a 1.80 | gain espéré pour  b 0.80 |
joueur  1 | gain espéré pour  a 1.00 | gain espéré pour  b 1.50 |
 a b
[{'a': 3, 'b': 3}, {'a': 3, 'b': 2}]

joueur  0 | gain espéré pour  a 1.50 | gain espéré pour  b 1.00 |
joueur  1 | gain espéré pour  a 1.20 | gain espéré pour  b 1.20 |
 a b
[{'a': 3, 'b': 4}, {'a': 4, 'b': 2}]

joueur  0 | gain espéré pour  a 1.29 | gain espéré pour  b 1.14 |
joueur  1 | gain espéré pour  a 1.33 | gain espéré pour  b 1.00 |
 a a
[{'a': 4, 'b': 4}, {'a': 5, 'b': 2}]

joueur  0 | gain espéré pour  a 1.50 | gain espéré pour  b 1.00 |
joueur  1 | gain espéré pour  a 1.43 | gain espéré pour  b 0.86 |
 a a
[{'a': 5, 'b': 4}, {'a': 6, 'b': 2}]

joueur  0 | gain espéré pour  a 1.67 | gain espéré pour  b 0.89 |
joueur  1 | gain espéré pour  a 1.50 | gain espéré pour  b 0.75 |
 a a
[{'a': 6, 'b': 4}, {'a': 7, 'b': 2}]

joueur  0 | gain espéré pour  a 1.80 | gain espéré pour  b 0.80 |
joueur  1 | gain espéré pour  a 1.56 | gain espéré pour  b 0.67 |
 a a
[{'a': 7, 'b': 4}, {'a': 8, 'b': 2}]

joueur  0 | gain espéré pour  a 1.91 | gain espéré pour  b 0.73 |
joueur  1 | gain espéré pour  a 1.60 | gain espéré pour  b 0.60 |
 a a
[{'a': 8, 'b': 4}, {'a': 9, 'b': 2}]

joueur  0 | gain espéré pour  a 2.00 | gain espéré pour  b 0.67 |
joueur  1 | gain espéré pour  a 1.64 | gain espéré pour  b 0.55 |
 a a
[{'a': 9, 'b': 4}, {'a': 10, 'b': 2}]

joueur  0 | gain espéré pour  a 2.08 | gain espéré pour  b 0.62 |
```

```
joueur  1 | gain espéré pour  a 1.67 | gain espéré pour  b 0.50 |
 a a
```

Quelques propriétés des dynamiques fictitious play

Supposons à présent que les deux agents utilisent une dynamique de fictitious play. On voit que l'on peut atteindre deux types de situations d'équilibre:

- des équilibres pour lesquels les probabilités sont concentrées sur une action unique (équilibre pur)
- des équilibres pour lesquels les probabilités sont réparties sur plusieurs actions (équilibre mixte)

On sait alors que:

- tout équilibre pur atteint par fictitious play doit être un équilibre de Nash
- la dynamique n'atteint pas nécessairement de situation d'équilibre (même mixte)
- il existe plusieurs catégories de jeux pour lesquels la convergence est garantie, par exemple les jeux à somme nulle.

4.3 Dynamique basée sur le regret

Appelons:

- score^t le score obtenu jusqu'à la période t par l'agent.
- $\text{score}^t(s)$ le score que l'agent aurait obtenu *s'il avait joué la stratégie* s de manière fixe (toutes choses égales par ailleurs, ie. si les autres agents avaient joué les mêmes stratégies à toutes les étapes)

Le **regret** de ne pas avoir joué s est alors, au moment t tout simplement

$$R^t(s) = score^t - score^t(s)$$

L'agent i choisit alors l'action (parmi l'ensemble d'actions S_i) selon une probabilité proportionnelle à son regret rapporté à la somme des regrets positifs. Autrement dit, la règle est simplement de choisir selon la propabilité donnée pour chaque action par:

$$\frac{R^t(s)}{\sum_{s' \in S_i} \max(0, R^t(s'))}$$

Illustration: Pierre-Feuille-Ciseaux

Dans le jeu Pierre-Feuille-Ciseaux (en anglais Rock-Paper-Scissor), aka chifoumi, on doit sélectionner une des trois actions parmi **rock, paper, scissor**. Il s'agit d'un jeu à somme nulle. La matrice des gains est donnée ci-dessous:

R \ B	**r**	**p**	**s**
r	(0,0)	(-1,1)	(1,-1)
p	(1,-1)	(0,0)	(-1,1)
s	(-1,1)	(1,-1)	(0,0)

Illustrons la dynamique de regret sur cet exemple, du point de vue de l'agent R:

coup R	**coup B**	**regret r**	**regret p**	**regret s**	**distrib**
r	p	0	1	2	(0,1/3,2/3)
s	s	1	0	2	(1/3,0,2/3)
s	s	2	-1	2	(1/2,0,1/2)
...	...	...	...	...	...

Cette dynamique peut être facilement implémentée. Ci-dessous je me base sur l'article:

> An introduction to counterfactual regret minimization. Todd Neller and Marc Lanctot. 2013.
> article (https://wwwf.imperial.ac.uk/~dturaev/neller-lanctot.pdf)

Ici on suppose que l'adversaire joue une stratégie mixte. Notre agent va calculer sa meilleure réponse à cette stratégie mixte en simulant l'algorithme de regret matching (self-play).

In [2…

```
import numpy as np
actions = [('r','p','s'),('r','p','s')]
matGains ={'r':{'r':(0,0),'p':(-1,1),'s':(1,-1)},\
           'p':{'r':(1,-1),'p':(0,0),'s':(-1,1)},\
           's':{'r':(-1,1),'p':(1,-1),'s':(0,0)}}
regret = {'r':0,'p':0,'s':0}
sumStrategy = {'r':0,'p':0,'s':0}
oppStrategy = {'r':0.4, 'p':0.3, 's':0.3} # on suppose que l'adve

def pickAction(strategy):
    #assert(sum(strategy.values())==1)
    r = np.random.uniform(0,1)
    cumulStrat = 0
    for s in strategy:
        cumulStrat+=strategy[s]
        if r < cumulStrat:
            return s
    return s

def cumulRegret(myAction,otherAction,regret):
    ''' accumule le regret pour chaque action, pour un choix spéc
    '''
    for action in actions[0]: #on met a jour le dico des regrets
        regret[action]+=matGains[action][otherAction][0]-matGains
    return

def cumulStrategy(sumStrategy, strategy):
    ''' accumule les stratégies (pour avoir statégie moyenne)
    '''
    for action in actions[0]:
        sumStrategy[action]+=strategy[action]
    return

def getAvgStrategy(sumStrategies):
    '''calcule la stratégie moyenne sur toutes les itérations
    '''
    total = sum(sumStrategies.values())
    assert(total>0)
    return {k:v/total for k,v in sumStrategies.items()}



def updateRegretBasedStrategy(regret):
    '''calcul de la strategie mixte proportionnelle aux regrets p
    on prend distribution uniforme si tous les regrets sont à 0
    '''
    posRegret = {'r':0,'p':0,'s':0}
    for action in posRegret:
```

```
        if regret[action]>0: # seulement si le regret est positif
            posRegret[action]=regret[action]
    total = sum(posRegret.values())
    if total==0:
        myStrategy = {k:1/len(actions[0]) for k in posRegret.keys
    else:
        myStrategy = {k:v/total for k,v in posRegret.items()}
    return myStrategy




def simulate(iterations,verbose=False):
    for step in range(iterations):
        otherAction = pickAction(oppStrategy)
        myStrategy = updateRegretBasedStrategy(regret)
        myAction = pickAction(myStrategy)
        cumulRegret(myAction,otherAction,regret) # calcul le regr
        cumulStrategy(sumStrategy,myStrategy)
        if verbose:
            print("my strategy is:", myStrategy)
            print("step",step, myAction,otherAction)
            print(regret)
    print("Strategy at final iteration        :", myStrategy)
    print("Avg strategy through all iterations:", getAvgStrategy(
    return


simulate(10000,False)
```

```
Strategy at final iteration        : {'r': 0.0, 'p': 1.0, 's': 0.
0}
Avg strategy through all iterations: {'r': 0.010152076787227963,
'p': 0.9868750211148699, 's': 0.002972902097902097}
```

On peut étudier la situation où les deux agents jouent en appliquant l'algorithme de regret matching. On constate qu'ils atteignent alors un équilibre (la stratégie moyenne jouée au cours de toutes les itérations converge).

In [2…

```
### convergence to equilibrium in RPS

regret = {'r':0,'p':0,'s':0}          # dico des regrets cumules
oppRegret = {'r':0,'p':0,'s':0}       # dico des regrets cumules
sumStrategy = {'r':0,'p':0,'s':0}     # dico des strategies cumul
sumOppStrategy = {'r':0,'p':0,'s':0}  # dico des strategies cumul


def simulateRMvsRM(iterations,verbose=False):
    for step in range(iterations):
        oppStrategy = updateRegretBasedStrategy(oppRegret)
        oppAction = pickAction(oppStrategy)
        myStrategy = updateRegretBasedStrategy(regret)
        myAction = pickAction(myStrategy)
        # cumul des regrets et strat des deux agents
        cumulRegret(myAction,oppAction,regret)
        cumulRegret(oppAction,myAction,oppRegret)
        cumulStrategy(sumStrategy,myStrategy)
        cumulStrategy(sumOppStrategy,oppStrategy)

        if verbose:
            print("my  strategy is:", myStrategy, "\nopp strategy
            print("step",step, myAction,oppAction)
            print("my regret:", regret, " opp regret:", oppRegret
    print("My strategy:         ", myStrategy)
    print("My average strategy: ", getAvgStrategy(sumStrategy))
    print("Opp strategy:        ", oppStrategy)
    print("Opp average strategy:", getAvgStrategy(sumOppStrategy)
    return

simulateRMvsRM(10000,False)
```

```
My strategy:         {'r': 0.0, 'p': 1.0, 's': 0.0}
My average strategy:  {'r': 0.3334084971099327, 'p': 0.3226294106
5081335, 's': 0.3439620922392539}
Opp strategy:        {'r': 1.0, 'p': 0.0, 's': 0.0}
Opp average strategy: {'r': 0.34350421266735914, 'p': 0.337562022
1610897, 's': 0.31893376517155114}
```

4.4 Rational Learning (Bayesian learning)

On note S l'ensemble des stratégies que l'agent x considère possible pour son adversaire, et H l'ensemble des histoires (séquences de coups) possibles. Il faut noter que contrairement aux approches

précédentes, les statégies envisagées peuvent être ici des stratégies non stationnaires (par exemple Tit-for-Tat, etc.).
Ainsi, étant donné une histoire h observée de coups de l'adversaire, la probabilité selon x que l'autre joueur joue s est donnée, selon application de la règle de Bayes, par :

$$P_x(s|h) = \frac{P_x(h|s)P_x(s)}{\sum_{s' \in S} P_x(h|s')P_x(s')}$$

Notez que les joueurs doivent avoir des croyances a priori sur la probabilité de l'autre de jouer certaines stratégies.

4. Le problème du bar de El-Farol

Considérons le problème suivant: chaque week-end des agents souhaitent se rendre dans un bar pour écouter de la musique. Les agents (supposons qu'ils sont au nombre de 100) ont la préférence suivante: ils souhaitent aller au bar pour écouter la musique, mais en même temps qu'il y ait trop de monde, sinon ils ne peuvent plus écouter correctement de la musique.

> Arthur, W. Brian (1994). "Inductive Reasoning and Bounded Rationality"

Plus précisément:

- si moins de 60 % de la population se rend au bar, l'agent préfère aller au bar
- si plus de 60% de la population se rend au bar, l'agent aurait préféré rester chez lui.

Ls agents se rendent (ou pas) le premier week-end au bar, ils observent la situation (ie. le nombre de personnes qui se sont rendues au bar effectivement), puis ils révisent leur stratégie pour le deuxième week-end, etc.

Il s'agit d'un problème connu sous le nom de **El-Farol problem**.

On s'interroge sur la règle de décision que peuvent utiliser les agents dans cette situation.

> Que se passe-t-il si tous les agents utilisent la même méthode déterministe pour décider s'ils vont au bar ou pas?

> Que se passe-t-il si tous les agents utilisent pile ou face pour décider s'ils vont au bar ou pas?

Arthur propose de faire reposer la décision des agents sur un **ensemble de prédicteurs**.

- prédicteurs cycliques: même nombre de personnes qu'il y a k semaines
- 100 - le nombre de personnes de la semaine dernière
- prédicteur fixe: 67 personnes
- moyenne glissante sur les 4 dernières semaines
- la tendance sur les 8 dernières semaines évaluée avec une régression linéaire et la méthode des moindre carrés

Chaque prédicteur se voit ensuite associé un score, selon la qualité de sa prédiction (+1 si la prédiction trop-de-monde / pas-trop-de-monde concorde avec l'observation).

Une bibliothèque de prédicteurs est créée, puis chaque agent reçoit un ensemble de prédicteurs qu'il aura à sa disposition. A chaque tour, les agents utiliseront leur prédicteur le plus performant jusqu'alors.

> Version du 17/03/2021. Style noTEXbook. **Rappel**: si la visualisation (en particulier des commandes LaTeX) est défaillante, ouvrir le même fichier dans le viewer jupyter (https://nbviewer.jupyter.org/github/nmaudet/teaching-iaro/blob/master/3-dynamiquesMultiagents/DynamiquesMultiagents.ipynb)

In […