HMM Doudou le hamster

Modélisation et Implémentation de HMM avec python

COULON Axel, Master AVR FST NANCY 2022-2023

28/12/2022

Sommaire

MODÉLISATION DU HMM

I - Définition du système

Un HMM se décrit formellement par :

- S : états possibles du système. Ici la variable que l'on cherche à prédire est la position de Doudou donc les états possibles sont : *Doudou sur la case S0, Doudou sur la case S1,….* Autrement dit, S={S0,S1,S2,S3,S4,S5,S6,S7}
- T : Matrice de transition. Cette matrice nous donne la probabilité de passer à un état suivant connaissant l'état à l'instant t. Autant dire que T représente la probabilité $P(S_{t+1}=s'|S_t=s)$. En prenant en compte les informations de l'énoncé, Lorsque Doudou est sur la case S0(représenté par 0 dans la matrice) à l'instant t, il a une probabilité 0.7 d'avancer sur la case S1 (n°1 dans la matrice), 0.1 d'aller dans l'autre sens et donc sur la case S7 et une probabilité de 0.2 de rester sur la case 0. On utilise la même logique pour chaque états et on obtient :

	0	1	2	3	4	5	6	7
0	0.2	0.1	0	0	0	0	0	0.7
1	0.7	0.2	0.1	0	0	0	0	0
2	0	0.7	0.2	0.1	0	0	0	0
3	0	0	0.7	0.2	0.1	0	0	0
4	0	0	0	0.7	0.2	0.1	0	0
5	0	0	0	0	0.7	0.2	0.1	0
6	0	0	0	0	0	0.7	0.2	0.1
7	0.1	0	0	0	0	0	0.7	0.2

Figure 1: Matrice de transition T

- Ω : Ensemble des observations possibles du système. Ici, on possède deux capteurs qui nous envoies une information similaire à un booléen (Détecté / Pas détecté). Les observations possibles sont alors Ω = {VV,VF,FV,FF} avec en index 1 le capteurs situé à la position S1 et en deuxième index, le capteur sur la case S4.
- O : fonction d'observation. On cherche à savoir la probabilité d'obtenir une observation donnée sachant un état connu à l'instant t. Autrement dit, cette fonction permet de prendre en compte le fait que les capteurs ne soit pas fiable à 100%. On calcule alors : $P(O_t=o|S_t=s)$. Par exemple, (les observations sont représentés par les lignes de la matrice dont l'index est respectifs à ceux de Ω) si Doudou est à l'état S0, les capteurs doivent renvoyer FF. La probabilité que les capteurs renvoient VV est 0.2*0.2 selon les données de l'énoncé. On garde la même logique pour les autres observations et les autres états et on obtient :

	0	1	2	3	4	5	6	7
0	0.04	0.18	0.04	0.04	0.18	0.04	0.04	0.04
1	0.16	0.72	0.16	0.16	0.02	0.16	0.16	0.16
2	0.16	0.02	0.16	0.16	0.72	0.16	0.16	0.16
3	0.64	0.08	0.64	0.64	0.08	0.64	0.64	0.64

Figure 2 : Matrice d'observation O

- Π_0 : Distribution initiale sur les états (t=0) : Selon l'énoncé, Π_0={1/8,1/8,1/8,1/8,1/8,1/8,1/8,1/8}

II - Étape de prédiction

On cherche ici à connaitre la distribution P(S1=s). On connait deux choses :

1. La distribution sur l'état S0 : $\Pi_0 = \{0.5,0.5,0,0,0,0,0,0\}$
2. La matrice de transition du système.

Or, il se trouve qu'on peut écrire la distribution de probabilité à l'état S1 comme le produit de celle à l'état S0 et la matrice de transition grâce à la loi de Bayes.

On a alors $P(S1 = s) = P(S1 = s|S0 = s') * P(S0 = s) = T * \Pi 0 =$ $\begin{matrix} 0.15 \rightarrow S0 \\ 0.45 \rightarrow S1 \\ 0.35 \rightarrow S2 \\ 0 \rightarrow S3 \\ 0 \rightarrow S4 \\ 0 \rightarrow S5 \\ 0 \rightarrow S6 \\ 0.05 \rightarrow S7 \end{matrix}$

III - Étape de correction

On nous donne l'information que l'observation est VF et on cherche à trouver P(S1=s|O1=VF). Nous on connait P(O1=VF|S1=s), P(S1=s) calculé à la question précédente. P(O1=VF) peut également être décomposé en éléments connus grâce à la loi de Bayes.

Formellement, on écrit d'après la loi de Bayes : $\mathrm{P}(\mathrm{S1} = \mathrm{s}|\mathrm{O1} = \mathrm{VF}) = \frac{\mathrm{P}(\mathrm{O1} = \mathrm{VF}|\mathrm{S1} = \mathrm{s}) * \mathrm{P}(\mathrm{S1}=\mathrm{s})}{P(O1=VF)}$

Avec $\mathrm{P}(\mathrm{O1} = \mathrm{VF}|\mathrm{S1} = \mathrm{s}) * \mathrm{P}(\mathrm{S1} = \mathrm{s}) =$

$$(0.16 \quad 0.72 \quad 0.16 \quad 0.16 \quad 0.02 \quad 0.16 \quad 0.16 \quad 0.16) * \begin{pmatrix} 0.15 \\ 0.45 \\ 0.35 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.05 \end{pmatrix} = (0.024 \quad 0.324 \quad 0.056 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0.008)$$

Et $P(O1 = VF) = \sum \mathrm{P}(\mathrm{O1} = \mathrm{VF}|\mathrm{S} = \mathrm{s}) * \mathrm{P}(\mathrm{S} = \mathrm{s}) = 0.024+0.324+0.056+0+0+0+0+0.008=0.412$

Donc $\mathrm{P}(\mathrm{S1} = \mathrm{s}|\mathrm{O1} = \mathrm{VF}) = \frac{\mathrm{P}(\mathrm{O1} = \mathrm{VF}|\mathrm{S1} = \mathrm{s}) * \mathrm{P}(\mathrm{S1}=\mathrm{s})}{P(O1=VF)} = \frac{1}{0.412} * (0.024 \quad 0.324 \quad 0.056 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0.008)$

$$= \begin{pmatrix} 0.058 \\ 0.786 \\ 0.136 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.019 \end{pmatrix}$$

PARTIE IMPLÉMENTATION

Dans cette partie, tout le code est retrouvable dans le fichier python en pièce jointe.

IV - Étape de prédiction implémentation et validation

```
def prediction(self,st):
    stPlus1=self.matriceTransition@st # MULTIPLICATION MATRICIELLE
    return stPlus1
```

```
(2c) Validation prédiction:  [0.15 0.45 0.35 0.   0.   0.   0.   0.05]
```

Figure 3 : Implémentation et validation de l'étape prédiction

V - Étape de correction implémentation et validation

```
def correction(self,stPlus1,observationTPlus1):
    if(observationTPlus1==self.observationsPossibles[0]):
        pObservationT1SachantEtatT1=self.matriceObservation[0,:]
    if(observationTPlus1==self.observationsPossibles[1]):
        pObservationT1SachantEtatT1=self.matriceObservation[1,:]
    if(observationTPlus1==self.observationsPossibles[2]):
        pObservationT1SachantEtatT1=self.matriceObservation[2,:]
    if(observationTPlus1==self.observationsPossibles[3]):
        pObservationT1SachantEtatT1=self.matriceObservation[3,:]
    pObservationTPlus1=sum(pObservationT1SachantEtatT1*stPlus1)
    pEtatT1SachantObservationT1=pObservationT1SachantEtatT1*stPlus1/pObservationTPlus1
    return pEtatT1SachantObservationT1
```

```
(2e) Validation correction:  [0.058 0.786 0.136 0.    0.    0.    0.    0.019]
```

Figure 4 : Étape de correction implémentation et validation

VI - Propagation des croyances

```
def propagation(self,st,observationTPlus1):
    stPlus1=self.prediction(st)
    pEtatT1SachantObservationT1=self.correction(stPlus1,observationTPlus1)
    return pEtatT1SachantObservationT1
```

Figure 5 : Implémentation de la méthode propagation

VII - Filtrage implémentation et vérification

```
def filtrage(self,st,observations : []):
    for n in range(len(observations)) :
        stn=self.propagation(st,observations[n])
        st=stn
    return stn
```

```
(2h) Validation Filtrage:  [0.025 0.496 0.117 0.344 0.007 0.    0.001 0.011]
```

Figure 6 : Implémentation et vérification du filtrage

On remarque ici une légère différence entre les valeurs que l'on obtient et celle supposées avoir. Ceci pourrait être expliquer par les arrondis que fait l'IDE que j'utilise (Spyder).

SIMULATION

```python
class Systeme:

    observation=""
    def __init__(self,HMM,distributionInitiale):
        self.HMM=HMM
        self.distributionCourante=distributionInitiale
        self.etatCourant=choices(self.HMM.etats,distributionInitiale)

    def evoluerEtatReel(self):
        nouveauEtatCourant=choices(self.HMM.etats,self.HMM.matriceTransition[:,self.etatCourant])
        self.etatCourant=nouveauEtatCourant
        self.observation=choices(self.HMM.observations(),self.HMM.matriceObservation[:,self.etatCourant])
        return

    def mettreAJourBelief(self):
        self.distributionCourante=self.HMM.propagation(self.distributionCourante,self.observation[0])
        return

    def evoluerSysteme(self):
        self.evoluerEtatReel()
        self.mettreAJourBelief()
        return
```

Figure 7 : Création du système HMM et des méthodes pour le faire évoluer

VIII - Main

```python
def mainSysteme(self):
    print("Etat initial avant evolution: ",self.etatCourant)
    for i in range(5):
        self.evoluerSysteme()
        print("Distribution Courante: ",self.distributionCourante)
        print("Observation Générée: ",self.observation[0])
        print("Nouveau Etat courant: ",self.etatCourant)
    return
```

Figure 8 : Création du main appelant les autres méthodes du système

```
(3e) Evolution du système:
Etat initial avant evolution:  [1]
Distribution Courante:  [0.24742268 0.09278351 0.57731959 0.         0.         0.
 0.         0.08247423]
Observation Générée:  00
Nouveau Etat courant:  [2]
Distribution Courante:  [0.14902737 0.03989449 0.23079459 0.51697989 0.         0.
 0.01055061 0.05275305]
Observation Générée:  01
Nouveau Etat courant:  [2]
Distribution Courante:  [0.12519516 0.02995637 0.22266645 0.46902952 0.08007821 0.00186771
 0.01307399 0.05813258]
Observation Générée:  00
Nouveau Etat courant:  [3]
Distribution Courante:  [0.03265858 0.00688398 0.05341402 0.12244698 0.73671206 0.0274354
 0.00462619 0.01582278]
Observation Générée:  01
Nouveau Etat courant:  [2]
Distribution Courante:  [0.02382938 0.00481561 0.03613767 0.17654525 0.038389   0.6794114
 0.0282788  0.01259289]
Observation Générée:  00
Nouveau Etat courant:  [3]
```

Figure 9 : Résultat du système pour t=0 à t=5

IX - Discussion de la simulation

Dans cette sous-partie, j'appellerai avancer tourner dans le sens horaire et reculer tourner dans le sens anti-horaire.

On voit que le système commence à l'état 1 puis il choisit d'avancer et de renvoyer l'observation 00. Sachant que la probabilité de ne pas avoir d'erreur sur l'observation des capteurs est plus grande que celle d'en avoir une, il est normal que le système pense que doudou est sur la case 2. Néanmoins doudou peut être aussi sur la case 0 mais la probabilité de reculer est bien plus faible que celle d'avancer. De plus, la probabilité que doudou reste sur S1 et que le capteur 1 se trompe est encore plus faible que la probabilité annoncé juste avant. Bien sûr, il existe également une possibilité où doudou a commencé sur la case 0 et a reculé mais ceci a évidemment une très faible probabilité dans le programme. Les résultats sont cohérents. Au 2ème pas du programme, Doudou choisit de rester sur la case 2 mais cette fois ci, le capteur sur la case 4 s'allume. Cependant, le programme comprend qu'il est impossible que doudou ait avancé de 2 cases en 1 pas de temps, donc la probabilité que doudou soit sur la case 4 est de 0 logiquement. La probabilité la plus forte reste que doudou ait tout simplement avancé et qu'on ait eu une erreur de capteur. C'est bien le cas dans notre système avec une probabilité d'être sur la case 3 de 0.5169. Il est intéressant de remarquer que le système donne une probabilité très faible que doudou ait reculé sur la case 1, c'est normal puisqu'il y a très peu de chance que doudou recule et que les 2 capteurs soient erronés en même temps. Au 3ème pas de temps, le système se corrige correctement. Au 4ème pas de temps, doudou recule mais l'observation retournée est 01 (catastrophe !). Forcément, le système pense que doudou est sur la case 4 car il y a de très grande chance d'avancer + d'allumer le capteur sur la case 4 par rapport au pas de temps précédent. Au dernier pas temps, le système ne se corrige pas car l'observation retournée est 00 donc il pense que doudou a tout simplement avancé sur la case 5 (ce qui est fait dans la plupart des cas). Le système pourra se corriger au prochain pas de temps si doudou avance et que l'observation générée est 01.

PARTIE RÉFLEXION - QUESTIONS OUVERTES

X - Estimation du sens de rotation

Ma première idée a été de dire qu'il faudrait créer un HMM qui se sert des observations que renvoi le nôtre. Formellement, notre HMM aurait été définit avec :

S={horraire,anti-horraire}

$$T = \begin{array}{c|cc} sens & H & AH \\ \hline H & 1 & 0 \\ AH & 0 & 1 \end{array}$$ car une fois le sens de rotation définit, il ne peut plus transiter.

Ω={[VF,FF,FF,FV],[VF,FF,FF,FF],[VV,FF,FF,FF]……..., ,[FF,FF,FV,FF]}. Il y a 2^8=256 observations différents possibles.

O= Matrice de 2 lignes (nombre états) et 2^8 colonnes, le nombre d'observations totales différentes.

Π_0={0.5,0.5} car on choisit le sens de rotation au hasard au début.

C'est la que je me suis rendu compte de mon erreur. Il est incohérent d'utiliser le sens de rotation comme états alors que c'est ce que l'on cherche. D'autant plus qu'on connaitrait le sens de rotation au temps t=0 grâce à la distribution initiale et puisque le sens ne peut pas changer après.

Ma deuxième idée a été de pensé que pour résoudre le problème dans notre HMM actuel, il fallait faire en sorte que les observations retournées au fut et à mesure de l'avancement étaient conservés et dès que l'on observait une des 2 suites d'observations suivante : [VF,FF,FF,FV] (sens horraire) ou [FF,FF,FV,FF,FF,VF] (sens AH) on pouvait conclure sur le sens de rotation de doudou.

Résultat :

```
Suivi du sens de rotation : ['10', '00', '00', '10', '01', '11', '00', '01', '00', '00', '00', '10', '00', '10',
'10', '10', '11', '00', '00', '00', '10', '10', '00', '00', '00', '00', '01', '10', '10', '01', '00', '00', '00',
'01', '00', '00', '00', '00', '01', '00', '10', '00', '10', '01', '00', '01', '11', '00', '00', '00']
```

Figure 10: Sens de rotation

On constate un problème : en 50 itérations on n'observe aucune des 2 suites d'observations ci-dessus. En y réfléchissant c'est normal puisque doudou peut changer de sens de rotation : la probabilité de faire demi-tour est de 0.1. Ajouté à cela le fait que les capteurs ne soient pas très fiables, il est quasiment impossible d'obtenir une des 2 suites représentantes des sens de rotation. Néanmoins on peut le déduire, doudou ira toujours dans le sens horaire puisqu'il a une probabilité d'avancer dans ce sens de 0.7. La chance donc, d'observe la suite d'observations représentatives du sens AH est infiniment petite.

Il faudrait donc modifier dans notre HMM la matrice de transition pour que ça marche mais on ne veut pas le faire ici car on veut utiliser le même HMM que celui dans la partie précédente.

X.a. - Modèle retenu

Finalement, la meilleure manière d'estimer le sens de rotation de doudou est de suivre notre algorithme et de considérer que doudou est vraiment à l'état estimé par l'algorithme. Avec cette méthode, on arrive rapidement à obtenir un sens de rotation :

```
def sensRotation(self):
    for i in range(50):
        self.evoluerSysteme()
        index_max=np.argmax(self.distributionCourante)
        if index_max==1:
            obs_parfaite='10'
        elif index_max==4:
            obs_parfaite='01'
        else:
            obs_parfaite='00'
        self.sens_rotation.append(obs_parfaite)
        l=len(self.sens_rotation)-1
        if(l>3):
            if (self.sens_rotation[l]=='01' and self.sens_rotation[l-1]=='00' and self.sens_rotation[l-2]=='00' and self
                print("Sens Horaire ! ",self.sens_rotation)
                break
            elif (self.sens_rotation[l]=='10' and self.sens_rotation[l-1]=='00' and self.sens_rotation[l-2]=='00' and se
                print("Sens Anti Horaire ! ",self.sens_rotation)
                break
```

```
Sens Horaire !  ['00', '00', '10', '00', '00',
'00', '10', '00', '00', '01']
```

Figure 11 : Sens de rotation : modèle retenu

XI - Estimation des probabilités des capteurs

XI.a. - Fonction d'observation à partir des données ?

Selon moi, Il est possible de déterminer la fonction d'observation à partir des données, à condition d'avoir bcp de données. Si les chercheurs connaissent le fonctionnement général de doudou, alors ils peuvent savoir le nombre de fois à peu près ou doudou va passer devant le capteur 1 par exemple. Si on fait une moyenne pondérée des mouvements possibles de doudou et leur probabilité respectives on obtient que doudou avance de 0.7*1+0.2*0+0.1*-1= 0.6 case par pas de temps. Donc en 100 pas de temps il est sensé avoir parcouru 60 cases et être passé 7.5 fois devant les 2 capteurs. A l'aide des observations, on peut déduire un nombre approximatif de faux positifs et de faux négatifs et ainsi déterminer une précision des capteurs bien qu'elle soit approximative. Exemple : On imagine qu'après 100 pas de temps doudou est passé 7 fois devant le capteur 1 et dans la réalité après 100 pas de temps on a relevé 5 valeurs 1 et 2 valeurs 0. On a comme précision du capteur 1 : 5/7=71,4% de précision.

XI.b. - Hypothèses

Oui c'est possible, dans ce cas-là il faut s'aider des données récupérées dans la question précédente et choisir le couple (x,y) qui se rapproche le plus des résultats qu'on a eu. Cependant on aura plutôt une liste de couples à la place d'un couple puisqu'il sera impossible de différencier les autres couples qui sont proportionnels à un couple choisi.

SÉQUENCE EXPLICATIVE

XII - Viterbi

On se sert de l'exemple du cours pour implémenter l'algorithme de viterbi.

```
"""Suite d'observations : 1,0 0,1 1,0 0,0"""
def viterbi(self, liste_observations, distribInitiale):
    meilleurParcours = np.zeros((liste_observations.shape[0]+1, len(distribInitiale)))
    meilleurParcours[0] = distribInitiale
    ligne = 0
    for observation in liste_observations:
        ligne += 1
        for colonne in range(len(distribInitiale)):
            for indice in range(len(distribInitiale)):
                meilleurParcours[ligne, colonne] += meilleurParcours[ligne - 1, indice]*buildFonctionTransition(self.
        for colonne in range(len(distribInitiale)):
            meilleurParcours[ligne, colonne] *= buildFonctionObservation(self.etats[colonne], observation)
    lienASuivre = np.argmax(meilleurParcours, axis=1)
    return lienASuivre
```

```
(5a)Viterbi pour la séquence:  [['10']
 ['01']
 ['10']
 ['00']]
Séquence d'états:  [0 1 2 1 2]
```

Figure 12 : Viterbi

On a bien le résultat escompté.