Artificial Intelligence

Week 4

Ramon Kits <m.s.kits@st.hanze.nl>, **Lisa-Lin Zuur** <l.f.m.zuur@st.hanze.nl>

BAYES RULE

Bayes rule speelt een belangrijke rol in AI (denk bijvoorbeeld aan Naive Bayes bij machine learning). In week 5 komt Bayes ook weer terug. Daarnaast is het goed om Bayes rule te begrijpen, want ook in de media en bij justitie[1] maken mensen denkfouten omdat ze niet de invloed van de z.g.n. a-priori kansen begrijpen.
Onderstaande casus komt uit het boek 'Thinking fast and slow' van Kahneman.

In de stad zijn twee taxibedrijven, de Groene en de Blauwe, actief in de stad. On de stad is afgelopen nacht een hit-and-run misdrijf gepleegd, en hierbij was een taxi betrokken.

Je krijgt de volgende gegevens: 85% van de taxi's in de stad zijn Groen en 15% zijn Blauw. Een getuige identificeerde de taxi als Blauw. De rechtbank testte de betrouwbaarheid van de getuige in dezelfde omstandigheden als de avond van het ongeval en concludeerde dat de getuige elk van de twee taxi's 80% van de tijd correct identificeerde, maar dus 20% van de tijd niet.

Wat is de waarschijnlijkheid dat de taxi die bij het ongeluk betrokken was Blauw was? Geef niet alleen het antwoord maar ook de berekening. Maak hierbij gebruik van de letters H en E, als volgt:

H = de taxi was blauw (de hypothese)
E = de getuige identificeerde de taxi als blauw (het bewijs of de test)

De Bayes rule luidt als volgt:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Dit is te lezen als: de kans op A gegeven B is gelijk aan de kans op B gegeven A keer de kans op A gedeeld door de kans op B. Deze laatste stap wordt vaak over het hoofd gezien, maar is essentieel voor de juiste uitkomst.

De berekening voor deze situatie volgt hieronder:

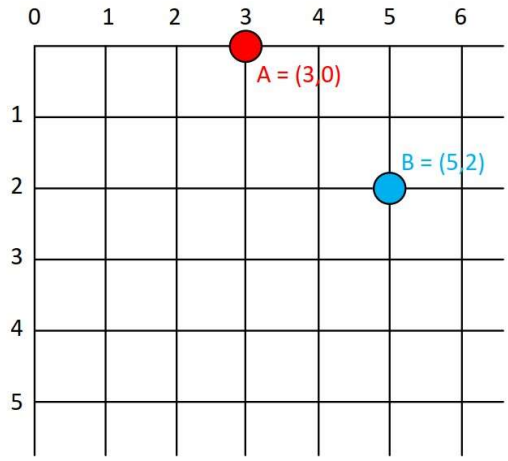
$$P(H|E) = \frac{P(E|H)P(H)}{P(E)} = \frac{0.8 * 0.15}{0.15 * 0.8 + 0.2 * 0.85} \approx 0.4138$$

[1] De zaak-Lucia de Berk is een beruchte gerechtelijke dwaling. Het gaat om een rechtszaak tegen de kinderverpleegkundige Lucia de Berk, die beschuldigd werd van meerdere moorden. Na 6,5 jaar onterechte detentie werd Lucia de Berk in 2010 vrijgesproken. Bij de veroordeling speelde statistische bewijs een grote rol. De econometrist Aart de Vos was de eerste die in de NRC op basis van Bayes rule aantoonde dat deze bewijsvoering niet juist was.

MARS ROBOT

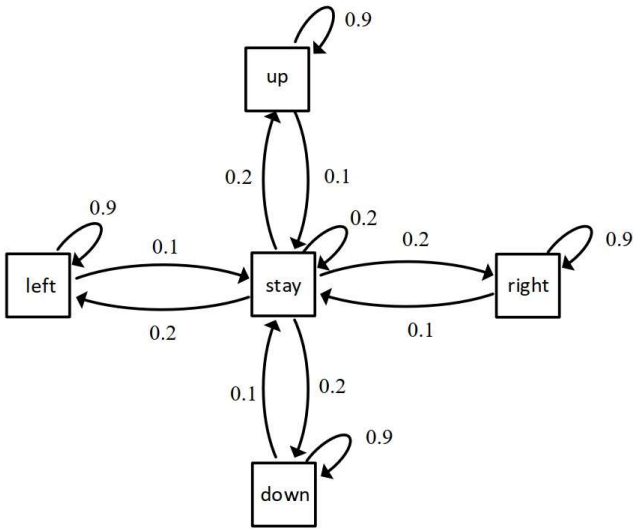
Stel dat een Mars-robot rondzwerft in een gebied dat gemodelleerd is als een rooster van 15 bij 15. De exacte locatie van de robot weten we niet, maar we krijgen wel onnauwkeurige metingen door via de sensoren op Mars. In deze opgaven gebruiken we een Hidden Markov Model om de bewegingen van de robot te modelleren.

De bewegingen van de robot zijn vrij voorspelbaar. Zie figuur 1. Bij elke tijdstap doet de robot één van de volgende vijf acties: hij blijft staan, gaat naar links of naar rechts, of gaat omhoog of naar beneden. De positie van de robot op tijdstip t hangt dus af van zijn positie op tijdstip t-1 *en van de actie* op tijdstip t-1.



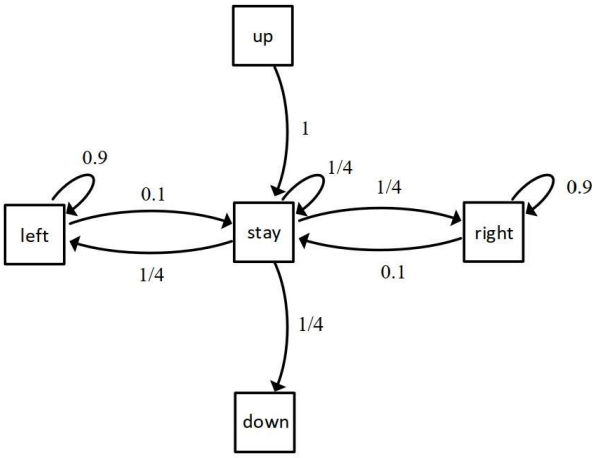
Figuur 1: Bewegingen van de robot op een rooster.
In positie A: L, R, D, S en in positie B: L, R, D, U, S. Zie ook figuur 2.

De acties van de van de robot zijn afhankelijk van de vorige actie. Zie het transitiediagram in figuur 2. Als we aannemen dat de robot zich *niet* op de grens van het rooster bevindt dan is zijn actie als volgt: als de vorige actie een beweging was (links, rechts, omhoog, omlaag), dan beweegt de robot weer in dezelfde richting met waarschijnlijkheid 0.9 en blijft staan met waarschijnlijkheid 0.1. Stond de robot stil, dan blijft hij weer staan met waarschijnlijkheid 0.2, of beweegt in een richting met waarschijnlijkheid 0.2. Merk op dat bijvoorbeeld (links, rechts, links) geen geldig pad is, maar (links, stay, rechts) wel.



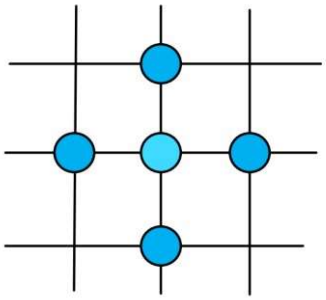
Figuur 2: Transitiediagram (robot bevindt zich niet op een grens)

De robot kan niet over de grens van het rooster heen. Wanneer de robot zich aan de bovenkant van het rooster bevindt, bijvoorbeeld op positie A = (3,0) is (zie figuur 1) dan is een actie ‘up’ niet mogelijk. Er blijven dan nog vier mogelijke acties over. In dat geval is het diagram in figuur 3 van toepassing: was de vorige actie van de robot ‘up’, dan blijft hij op A. Verder kan de vorige actie niet ‘down’ geweest zijn, want dan zou de robot zich buiten het rooster begeven. Eenzelfde model en redenering geldt ook voor de andere grenzen. Zie figuur 3.



Figuur 3: Transitiediagram (robot bevindt zich in positie A van figuur 1)

De exacte positie van de robot is onbekend. We hebben alleen toegang tot metingen van onnauwkeurige sensoren die in berichten naar de aarde worden verzonden. Wanneer de robot zich in een positie P bevindt, dan kunnen de sensoren P doorgeven, maar ook één van de buren van P. Deze kansen zijn uniform verdeeld, dus elk van deze mogelijke metingen (waarnemingen) hebben een kans van 1/5. Bevindt de robot zich in (7,7) dan is er een kans van 0.2 dat de sensoren (7,8) doorgeven. Bevindt de robot zich op de grens, dan geven de sensoren alleen posities door binnen het rooster (er blijven dan vier i.p.v. vijf posities over). Zie figuur 4.



Figuur 4: Wanneer de werkelijke positie in het centrum is, dan zijn er vijf mogelijke waarnemingen met gelijke kansen.

Op Blackboard is de startcode te vinden. Wanneer op de startknop wordt gedrukt dan zal de robot een geheel willekeurig pad uitvoeren. Daarnaast bevat model.py al enkele functies die nodig zijn voor het verder implementeren van het programma.

Bij de opgaven (a) t/m (h) mag je aannemen dat het rooster oneindig groot is, dus dat er geen problemen zijn met de randen van het bord.

a) De functie transition_model is al gegeven. Gebruik deze functie om de robot 100 stappen door het rooster te laten maken. Je mag hierbij aannemen dat de startpositie van de robot (7,7) is met (vorige) actie ‘stay’.

b) Wat is de kans dat een *willekeurige* move 'left' is? Wat is de kans dat een *willekeurige* move 'stay' is?

Om de tijd te berekenen dat het model gemiddeld in de toestand ‘stay’ zit, is het nodig om eerst de steady-state vector te berekenen. Deze steady-state vector wordt ook wel geschreven als π. Deze vector is te berekenen door de volgende formule: $\pi = \pi P$. Hierbij is π de steady-state vector en P de transitie matrix. In het geval van dit model is dit de volgende matrix:

$$P = \begin{bmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.1 & 0.9 & 0 & 0 & 0 \\ 0.1 & 0 & 0.9 & 0 & 0 \\ 0.1 & 0 & 0 & 0.9 & 0 \\ 0.1 & 0 & 0 & 0 & 0.9 \end{bmatrix}$$

Voor de volgorde van de toestanden is gekozen voor de volgende volgorde:

$$S = [stay \quad left \quad right \quad up \quad down]$$

Vervolgend moet de matrix π berekend worden.

$$\pi = [\pi_{stay} \quad \pi_{left} \quad \pi_{right} \quad \pi_{up} \quad \pi_{down}] \begin{bmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.1 & 0.9 & 0 & 0 & 0 \\ 0.1 & 0 & 0.9 & 0 & 0 \\ 0.1 & 0 & 0 & 0.9 & 0 \\ 0.1 & 0 & 0 & 0 & 0.9 \end{bmatrix}$$

Deze vergelijking kan opgelost worden met behulp van eigenwaarden en eigenvectoren. Hiervoor is gebruik gemaakt van de Python library numpy met de functie `linalg.eig`. Hieruit volgen meerder eigenwaarden en eigenvectoren. De eigenwaarde die gebruikt moet worden is de eigenwaarde met de waarde 1. Deze eigenwaarde heeft de volgende genormaliseerde eigenvector:

$$\pi = [0.11111111 \quad 0.22222222 \quad 0.22222222 \quad 0.22222222 \quad 0.22222222]$$

Deze vector geeft de kans weer dat het model in een bepaalde toestand zit. De kans dat het model in de toestand ‘stay’ zit is dus 0.11111111. Dit betekent dat het model gemiddeld ongeveer 11.11% van de tijd in de toestand ‘stay’ zit en ongeveer 22.22 van de tijd in de toestand ‘left’.
Dit is ook logisch te beredeneren. Als we ‘stay’ als de begin toestand gebruiken, dan is de kans dat het model in de toestand ‘stay’ blijft 0.2. De kans dat het model naar een andere toestand gaat is 0.8, welke in de volgende stap een kans van 0.1 heeft om weer terug te gaan naar de toestand ‘stay’. Dit doet terecht vermoeden dat de toestand ‘stay’ slechts de helft zo vaak voorkomt als ieder van de andere toestanden. Gezien de kans op de andere toestanden gelijk moet zijn, zal de kans op ‘stay’ dus 0.11111111 zijn. Dit is immers de enige manier waarop de toestand ‘stay’ slechts de helft zo vaak voorkomt als ieder van de andere toestanden.

c) Stel dat de robot in de toestand ‘stay’ zit. Wat is de kans dat de robot *precies* 3 keer (dus niet 4 keer) een ‘stay’ achter elkaar uitvoert?

De robot staat reeds in de toestand 'stay' dus om deze precies 3 keer uit te voeren zal deze toestand hierna nog 2 keer moeten volgen. Dit leidt tot de volgende berekening:

$$1 \cdot 0.2 \cdot 0.2 = 0.04$$

d) Wat is de gemiddelde tijdsduur dat het model in de toestand 'stay' zit. Dus als de robot oneindig lang door zou lopen, hoe lang zit de robot dan gemiddeld in de toestand 'stay'?

Elke move duurt 40 milliseconden. De kans dat deze de eerstvolgende move niet meer 'stay' is, is 0.8. In 20 van de gevallen is de toestand wederom 'stay', etcetera. Deze kansen vormen een geometrische reeks. De verwachte tijd dat het model in de toestand 'stay' zit is dus $0.2^0 \cdot 40 + 0.2^1 \cdot 40 + 0.2^2 \cdot 40 + \cdots + 0.2^\infty \cdot 40$. Dit kan herschreven worden naar een som van een geometrische reeks: $\sum_{i=0}^{\infty} 40 \cdot 0.2^i = 50$. Zo te zien convergeert deze reeks naar 50. Dit betekent dat het model gemiddeld 50 milliseconden in de toestand 'stay' zit. Deze berekening is tevens te herschrijven als $40 \cdot \frac{1}{1-0.2} = 50$.

e) Hoeveel mogelijk paden zijn er van 10 stappen? Merk op (1) dat een 'stay' niets toevoegt aan een pad en (2) als je bijvoorbeeld een 'left' hebt gedaan, de volgende actie een 'left' of een 'stay' moet zijn. Schrijf hiervoor een kort (recursief) programma dat het aantal paden telt.

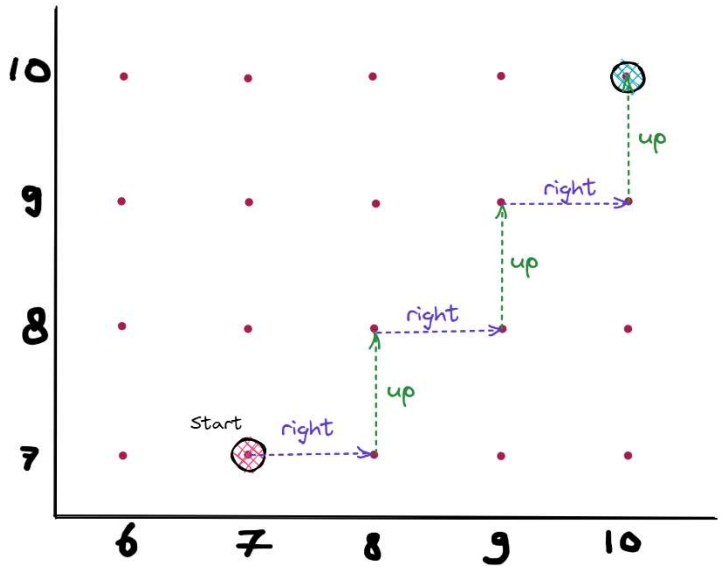
```python
possibilities: dict[str, list[str]] = {
    'Stay': ['Left', 'Up', 'Down', 'Right'],
    'Left': ['Left', 'Stay'],
    'Right': ['Right', 'Stay'],
    'Up': ['Up', 'Stay'],
    'Down': ['Down', 'Stay']
}


def find_path_count(state: str, depth: int) -> int:
    if depth == 0:
        return 1
    count = 1 if state != 'Stay' else 0
    for next_state in possibilities[state]:
        count += find_path_count(next_state, depth - 1)
    return count
```

Met behulp van deze recursieve functie komen er 23964 mogelijke paden uit.

f) Het doorlopen van alle paden bij (e) kan je zien als het doorlopen van een boom. Hoe groot is dan de (gemiddelde) branching factor van deze boom? (Ik kom op 2.19).

Vanaf een toestand anders dan ‘stay’ is de branching factor 2. Vanaf de toestand ‘stay’ is de branching factor 5. De kans op toestand ‘stay’ is 0.11111111. De kans op een andere toestand is 0.88888889. De verwachte branching factor is dus $2 \cdot 0.88888889 + 5 \cdot 0.11111111 = 2.33333333$.

g) Hoeveel paden zijn er van (7,7) naar (10,10) met de beperking dat een pad precies 6 stappen lang is?



In figuur 2 is een mogelijk pad te vinden tussen (7,7) en (10, 10). Dit pad bestaat uit [“Right”, “Up”, “Right”, “Up”, “Right”, “Up”]. Alle permutaties van deze volgorde gaan ook valide paden zijn; alle antwoorden die een andere richting dan “Right” of “Up” hebben gaan invalide zijn, aangezien antwoorden met 3 keer “Right” en 3 keer “Up” altijd over zes stappen bij (10, 10) zijn.

Om alle optimale paden te vinden zou je dus de faculteit aantal stappen kunnen delen door de faculteit van de horizontale verschil keer de faculteit van de verticale verplaatsing.

$$\frac{n!}{v! \cdot h!}$$

Om deze toe te passen bij deze opgave komt men op het volgende:

$$\frac{6!}{3! \cdot 3!} = 20$$

h) Hoeveel paden zijn er van (x1, y1) naar (x2, y2) met de beperking dat de lengte van het pad gelijk is aan de Manhattan afstand (= het kortste pad)?

Terugkomend op opgave G, zijn er

$$\frac{n!}{v! \cdot h!}$$

paden als

$$v = x_1 - y_1$$

en

$$h = x_2 - y_2$$

We ontvangen nu berichten van de sensoren, zodat we een schatting kunnen gaan maken van het werkelijke pad dat de robot aflegt met het Viterbi-algoritme. We nemen aan dat initieel elke positie even waarschijnlijk is en dat de initiële actie van de robot ‘stay’ is. Je ziet

in de code dat een toestand een combinatie is van positie en vorige actie. Als we als voorbeeld positie (5,4) waarnemen dan kan zijn er maar vijf mogelijke posities waar de robot zich op dat moment kan bevinden. Als de vorige acte een ‘links’ was, dan kan de huidige actie alleen maar een ‘links’ of een ‘stay’ zijn. Omdat we alle deze kansen weten gegeven een reeks van waarnemingen kunnen we ook het meest waarschijnlijke pad berekenen.

i) Als er een waarneming (5,5) is, is wat is de kans dat dit ook juist is?

Aangezien de kansen uniform verdeeld zijn en er 5 mogelijke states zijn waaronder de juiste, is de kans dat de waarneming correct is 1 op 5, dus 0,2