

Algorithme des k plus proches voisins. Habitat des renards.



FIGURE 1 – zones habitées par les renards

Le renard est un animal qui peut habiter dans plusieurs types d'habitats pouvant être des plaines, des montagnes, des environnements ruraux, périurbains, voire urbains. La loi Biodiversité 2016 ainsi que l'arrêté du 3 août 2023 prévoient que le renard n'est plus un animal "Nuisible" mais "Susceptible d'être nuisible".

Malgré la loi, le renard est souvent chassé des zones où il pourrait normalement évoluer et réguler la faune. Pour éviter certaines dérives, il est recommandé de surveiller la population de renards dans les zones concernées et donc prédire si un renard peut habiter cette zone.

Pour la prédiction des zones habitables d'un renard, on considère les caractéristiques suivantes :

- la végétation • la proximité de l'eau • la densité urbaine • la disponibilité de proies

Ces caractéristiques sont toutes mesurées sur une échelle de 1 à 10.

De plus, on dispose pour les zones connues d'une caractéristique supplémentaire indiquant, par un booléen, la présence d'un renard.

```
1 zones = [  
2     {'vegetation': 9, 'proximite_eau': 6, 'densite_urbaine': 0,  
3       'disponibilite_proies': 4, 'presence_renard': True},  
4     {'vegetation': 10, 'proximite_eau': 5, 'densite_urbaine': 9,  
5       'disponibilite_proies': 10, 'presence_renard': False},  
6     {'vegetation': 8, 'proximite_eau': 5, 'densite_urbaine': 1,  
7       'disponibilite_proies': 6, 'presence_renard': False},  
8     {'vegetation': 8, 'proximite_eau': 5, 'densite_urbaine': 7,  
9       'disponibilite_proies': 6, 'presence_renard': True},  
10    {'vegetation': 6, 'proximite_eau': 9, 'densite_urbaine': 9,  
11      'disponibilite_proies': 6, 'presence_renard': True},  
12    {'vegetation': 5, 'proximite_eau': 3, 'densite_urbaine': 2,  
13      'disponibilite_proies': 9, 'presence_renard': False}  
14    ]
```

On évalue la distance euclidienne entre 2 habitats de végétation, de proximité de l'eau à l'aide de la loi :

$$d = \sqrt{(v' - v)^2 + (p' - p)^2}$$

Ce que l'on peut traduire en langage python par :

```

1 h1 = {'vegetation': 9, 'proximite_eau': 6, 'densite_urbaine': 0,
      'disponibilite_proies': 4, 'presence_renard': True},
2 h2 = {'vegetation': 10, 'proximite_eau': 5, 'densite_urbaine': 9,
      'disponibilite_proies': 10, 'presence_renard': False}
3 v1 = h1['vegetation']
4 v2 = h2['vegetation']
5 p1 = h1['proximite_eau']
6 p2 = h2['proximite_eau']
7 d = ((v2-v1)**2+(p2-p1)**2)**0.5

```

Pour un habitat h de végétation v, de proximité de l'eau p, de densité urbaine u et de disponibilité des proies d et h un habitat ayant, de même, les caractéristiques v, p, u, d, cette loi de distance devra aussi prendre en compte u, d, u', d'.

Question 1 : Modifier le code de la fonction `distance` qui prend en paramètres deux habitats sous la forme de dictionnaires contenant au moins les clés 'vegetation', 'proximite_eau', 'densite_urbaine', 'disponibilite_proies' et qui renvoie la distance entre ces deux habitats selon la formule présentée au-dessus, mais adaptée.

On étudie un nouvel habitat, appelé nouveau, pour savoir si celui-ci est propice à la présence d'un renard.

```

1 nouveau = {'vegetation': 5, 'proximite_eau': 2,
2           'densite_urbaine': 4, 'disponibilite_proies': 6}

```

Question 2 : Utiliser la fonction `distance` pour calculer la distance de `nouveau` avec le premier habitat du tableau `zones`. Placer alors un nouveau couple 'distance':valeur dans le dictionnaire de ce premier habitat. On veut obtenir pour `zones[0]` :

```

1 {'vegetation': 9, 'proximite_eau': 6, 'densite_urbaine': 0, '
  disponibilite_proies': 4, 'presence_renard': True, 'distance':
  7.211102550927978}

```

Question 3 : Compléter le code de la fonction `distance_d_un_habitat` qui prend en paramètres un `habitat` sous la forme de dictionnaire et une liste d'habitats sous la forme de liste de dictionnaires. La fonction doit ajouter pour chaque dictionnaire des habitats la clé `distance`, et la valeur calculée par rapport à l'habitat. Utilisez votre fonction pour ajouter à chaque dictionnaire d'habitat la distance avec `nouveau`.

On rappelle la notion de *tri* d'une table : Une table est triée selon un attribut (une colonne, ou bien une clé d'un dictionnaire), si les lignes sont ordonnées selon la valeur croissante de cet attribut.

Par exemple, la table suivante ne présente aucun ordre particulier, elle n'est pas *triée* :

```
1 [{ 'vegetation': 9, 'presence_renard': True, 'distance':  
    7.211102550927978},  
2 { 'vegetation': 10, 'presence_renard': False, 'distance':  
    8.660254037844387},  
3 { 'vegetation': 8, 'presence_renard': False, 'distance':  
    5.196152422706632}]
```

Alors que la table suivante est *triée* selon la clé distance :

```
1 [{ 'vegetation': 8, 'presence_renard': False, 'distance':  
    5.196152422706632},  
2 { 'vegetation': 9, 'presence_renard': True, 'distance':  
    7.211102550927978},  
3 { 'vegetation': 10, 'presence_renard': False, 'distance':  
    8.660254037844387}]
```

La fonction trier donnée dans le script permet de réaliser ce tri. Elle prend en paramètre un tableau (une liste de dictionnaires), et une cle de tri (une clé qui doit être présente dans chaque dictionnaire).

Question 4 : Utiliser cette fonction pour trier le tableau `zones` à partir de la clé `distance`. Vérifier que le tableau est alors bien trié. Observer enfin si le premier habitat du tableau `zones` montre la présence d'un renard.

La méthode statistique de recherche des *k plus proches* voisins est une méthode de prediction. On cherche à déterminer la **classe** d'un nouvel objet en la comparant à celle des éléments du tableau qui sont les plus **semblables**.

La méthode des *k plus proches* voisins suit les opérations suivantes :

1. calculer la distance euclidienne du nouvel objet avec chaque élément du tableau
2. écrire cette distance dans chaque ligne, pour chaque élément du tableau
3. trier le tableau par distance croissante
4. réduire le tableau à ses k premières lignes. Ce sont les k plus proches voisins.
5. évaluer la proportion d'éléments qui ont la valeur True pour la classe considérée.

Dans notre travail, ce sera la classe `presence_renard` que l'on va analyser pour les k premiers éléments du tableau trié.

Question 5 : Ecrire une fonction `presence_renard(k, habitat, habitats)` qui doit exécuter cette procédure, et retourner `True` si, pour les k habitats les plus proches, il y a plus souvent présence d'un renard. `False` sinon.

Par exemple, avec les données suivantes, on aura `False` :

```
1 nouveau = { 'vegetation': 5, 'proximite_eau': 2,  
2             'densite_urbaine': 4, 'disponibilite_proies': 6}  
3 zones = [{ 'vegetation': 9, 'presence_renard': True},  
4 { 'vegetation': 10, 'presence_renard': False},
```

```
5 {'vegetation': 8, 'presence_renard': False}]  
6 print(presence_renard(3, nouveau, zones))  
7 > False
```

Question 6 : Tester votre fonction sur le tableau **zones** fourni dans le script. Puis testez le sur le fichier **donnees_habitat.py** fourni dans le dossier.

Partie 2

Contenu du projet

- énoncé pdf
- fichier de travail `prediction_habitat.py`
- fichier de données `donnees_habitat.py`

Partie 3

corrigé

```
from math import sqrt
from donnees_habitats import zones_connues

nouveau = {'vegetation': 5, 'proximite_eau': 2,
            'densite_urbaine': 4, 'disponibilite_proies': 6}

zones = [
    {'vegetation': 9, 'proximite_eau': 6, 'densite_urbaine': 0,
     'disponibilite_proies': 4, 'presence_renard': True},
    {'vegetation': 10, 'proximite_eau': 5, 'densite_urbaine': 9,
     'disponibilite_proies': 10, 'presence_renard': False},
    {'vegetation': 8, 'proximite_eau': 5, 'densite_urbaine': 1,
     'disponibilite_proies': 6, 'presence_renard': False},
    {'vegetation': 8, 'proximite_eau': 5, 'densite_urbaine': 7,
     'disponibilite_proies': 6, 'presence_renard': True},
    {'vegetation': 6, 'proximite_eau': 9, 'densite_urbaine': 9,
     'disponibilite_proies': 6, 'presence_renard': True},
    {'vegetation': 5, 'proximite_eau': 3, 'densite_urbaine': 2,
     'disponibilite_proies': 9, 'presence_renard': False}
]

def distance(h1, h2):
    """
    Calcule la distance euclidienne entre deux habitats.
    entrée :
        - habitat_1 : dictionnaire représentant un habitat.
        - habitat_2 : dictionnaire représentant un autre habitat.
    sortie :
        - float : distance euclidienne entre habitat_1 et
        habitat_2.
    """
    v1 = h1['vegetation']
    v2 = h2['vegetation']
    p1 = h1['proximite_eau']
    p2 = h2['proximite_eau']
    de1 = h1['densite_urbaine']
    de2 = h2['densite_urbaine']
    di1 = h1['disponibilite_proies']
    di2 = h2['disponibilite_proies']
    d = ((v2-v1)**2+(p2-p1)**2+(de2-de1)**2+(di2-di1)**2)**0.5
    return d

def distance_d_un_habitat(habitat, habitats):
    """
    Calcule la distance entre un habitat et chaque habitat de la
    liste.
```

```
47     Et ajoute un couple 'distance':valeur
48     où valeur est la distance entre habitat et chaque habitat de
    la liste.
49     entrée :
50         - habitat : dictionnaire représentant un habitat.
51         - habitats : liste de dictionnaires représentant des
    habitats.
52     sortie :
53         - None: la liste est modifiée en place
54     '''
55     for h in habitats:
56         h['distance'] = distance(nouveau,h)
57
58
59
60 def trier(tableau, cle):
61     '''
62     trie le tableau selon l'une de ses clés
63     entrée :
64         - tableau : liste de dictionnaires représentants les
    habitats.
65         - cle : une des clés commune à chaque dictionnaire.
66     sortie :
67         - None: la liste est triée en place
68     '''
69     tableau.sort(key=lambda x:x[cle])
70
71
72
73 def presence_renard(k, habitat, habitats):
74     '''
75     Vérifie si l'habitat donné a plus de k/2 voisins avec des
    renards.
76     entrée :
77         - k : entier représentant le nombre d'habitats à considé
    rer.
78         - habitat : dictionnaire représentant un habitat.
79         - habitats : liste de dictionnaires représentant des
    habitats.
80     sortie :
81         - bool : True si l'habitat a plus de k/2 voisins avec des
    renards, False sinon.
82     '''
83     k_plus_proches(k, habitat, habitats)
84     n_renards = 0
85     for h in habitats[:k+1]:
86         if h['presence_renard']:
87             n_renards += 1
88     return n_renards > k/2
```