I61_DAUSSY_2023

DAUSSY Fabio

April 2023

1 TP I61

Dans la première partie de ce TP nous allons dans un premier temps étudier un algorithme qui va nous permettre de calculer l'entropie d'une séquence, soit une entropie conjointe de n éléments où n représente la longueur de la séquence, en utilisante la règle de la chaine.

Pour cela, nous allons réalisé les étapes suivantes :

- Construire l'**hypercube** de la matrice d'entrée
- Écrire un algorithme qui va **projeter** cet hypercube dans des dimensions inférieures (pour calculer des probabilités jointes avec des variables aléatoire en moins).
- **Tester** le code pour voir si tout cela fonctionne.

Voici le code qui permet de construire l'hypercube :

```
def hypercube(OBS):
    """ Construit l'hypercube de la matrice """

    r = len(OBS) # On recupere le nombre de realisation

    dico = {} # On initialise le dictionnaire, dont les cles sont les sequences
    # Ce dictionnaire va compter les occurences des sequences.

    # Pour chaque sequence, on ajoute sa concatenation en cle du dico si
    # elle n'y est pas, sinon on ajoute 1 a la cle existente.
    for i in range(r):

        if tuple(OBS[i]) not in dico.keys(): # On transtype en tuple car pas de liste
        # en cle de dictionnaire
            dico[tuple(OBS[i])] = 1
        else:
            dico[tuple(OBS[i])] += 1
```

```
# On calcule ensuite les probabilités jointes de nos sequences et on les stocke dans
# le dictionnaire
for k in dico.keys():
    dico[k] = dico[k] / r
return dico
```

Grâce à l'hypercube, il est très facile de calculer l'entropie jointe de notre séquence et ses réalisations :

```
def entropie_jointe(OBS):
"""
Calcule l'entropie jointe de la séquence et des ses réalisations, stockée dans
la matrice OBS
"""
somme = 0
hcube = hypercube(OBS)

for key in hcube.keys():

    somme += hcube[key] * log2(hcube[key])

return - somme
```

Ensuite, il faut implanter le code qui permet de réduire l'hypercube pour pouvoir calculer les entropies conditionnelles :

```
def reduire(hypercube):
"""
Cette fonction va permettre de réduire l'hypercube de une dimension
"""

nouveau = {} # Le nouveau dictionnaire qui va stocker les probabilites jointes
# de hypercube avec une dimension de moins


for key in hypercube.keys():
    prob = hypercube[key] # On memorise la probabilite courante
    # On regarde si la nouvelle cle existe dans le nouveau dico
    if key[:-1] not in nouveau.keys():
        # si elle n'y est pas on l'ajoute
        nouveau[key[:-1]] = prob # avec la probabilite memorisee !

    else:
        # si elle y est on fait la somme des probabilites
        nouveau[key[:-1]] += prob

return nouveau
```

2 TP 2 I61

Aujourd'hui nous allons implanter les calculs suivants :

- $IM(X_i; Y_i) = H(X_i) + H(Y_i) - H(X_i, Y_i)$
- $IM(X^N; Y_1) = H(X^N) + H(Y_1) - H(X^N, Y_1)$
- $IM(X^N \to Y^N) = \sum_{n=1}^{N} IM(X^n; Y_n | Y^{n-1})$

Pour le premier calcul:

$$IM(X_i; Y_i) = H(X_i) + H(Y_i) - H(X_i, Y_i) \tag{1}$$

On calcule l'entropie de $H(X_i)$ d'une variable aleatoire parmis une séquence grace aux fonctions suivantes:

```
def probabilite(OBSXN, i):
    """
    Calcule les probabilité d'apparition de la variable aléatoire en
    Xi pour toutes les realisations
    """
    r = len(OBSXN) # Le nombre de realisations
    dico = {} # Le dictionnaire qui va contenir les probas

    # On compte les occurences
    for ligne in range(r):
        if OBSXN[ligne][i] not in dico.keys():
            dico[OBSXN[ligne][i]] = 1
        else:
            dico[OBSXN[ligne][i]] += 1

    # On calcule les probas
    for k in dico.keys():
        dico[k] = dico[k] / r

    return dico

def entropie(OBSXN, i):
    """
    Calcule l'entropie de la variable aléatoire Xi
    """

    proba = probabilite(OBSXN, i)
    somme = 0 # La somme de l'entropie

    for k in proba.keys():
```

```
        somme += proba[k] * log2(proba[k])

    return - somme
```

Ensuite on calcule la probabilité jointes de X_i et Y_i comme suit :

```
def construire_hypercube_dim2(XN, YN, i):
    """
    Construit l'hypercube des probabilités jointes des variables aléatoire Xi et Yi
    """

    dico = {} # hypercube

    r = len(XN)

    for ligne in range(r):
        # On a bien comme cle XiYi
        cle = (XN[ligne][i],YN[ligne][i])
        print(cle)

        if cle not in dico.keys():

            dico[cle] = 1

        else:

            dico[cle] += 1

    # Calcule des probas
    for k in dico.keys():

        dico[k] /= r

    return dico
```

On calcule l'entropie via l'hypercube :

```
def entropie_via_hypercube(hypercube):
    """ Calcule la formule de l'entropie jointe uniquement via l'hypercube"""

    somme = 0

    for key in hypercube.keys():
        somme += hypercube[key] * log2(hypercube[key])
```

```
    # pour éviter de retourner -0.0
    if somme == 0:
        return 0

    return -somme
```

Et grâce à ses fonctions on peut calculer la formule initiale :

$$IM(X_i; Y_i) = H(X_i) + H(Y_i) - H(X_i, Y_i) \tag{2}$$

```
def IM(OBSXN, OBSYN, i):
    """
    Calcule l'information mutuelle entre les variables à l'indice i de
    OBSXN et OBSYN
    """

    concat = construire_hypercube_dim2(OBSXN, OBSYN, i)

    res = entropie(OBSXN,i) + entropie(OBSYN, i) - entropie_via_hypercube( concat )

    return res
```

Il est alors simpliste de calculer

$$IM(X^N; Y_1) = H(X^N) + H(Y_1) - H(X^N, Y_1) \tag{3}$$

Pour cela il faut calculer l'hypercube de X^N et son entropie (cf. fonction hypercube et fonction entropie_via_hypercube)

Ce qui nous donne :

```
def hypercube_XNYi(OBSXN, OBSYN, i):
    """
    Calcule l'hypercube de XN et Yi concaténé ainsi que les probabilités
    """

    r = len(OBSXN)
    dico = {}

    for ligne in range(r):

        cle = tuple(OBSXN[ligne] + [OBSYN[ligne][i]])

        if cle not in dico.keys():

            dico[cle] = 1
```

```
        else:
            dico[cle] += 1

    for k in dico.keys():
        dico[k] /= r

    return dico


def IM2(OBSXN, OBSYN, i):
    """
    Calcule l'information mutuelle entre les variables XN et Yi
    """

    hcubeXNYi = hypercube_XNYi(OBSXN, OBSYN, i)
    hcubeXN = hypercube(OBSXN, len(OBSXN[0]))

    res = entropie_via_hypercube(hcubeXN) + entropie(OBSYN, i)
        - entropie_via_hypercube(hcubeXNYi)

    return res
```

Enfin, pour calculer l'information mutuelle dirigée, on développe la formule:

$$IM(X^N \to Y^N) = \sum_{n=1}^{N} IM(X^n; Y_n | Y^{n-1}) \tag{4}$$

avec

$$IM(X^n; Y_n | Y^{n-1}) = H(X^n, Y^{n-1}) - H(Y^{n-1}) + H(Y^n) - H(X^n, Y^n) \tag{5}$$

Ainsi, on obtient :

```
def construire_hypercube_concat(XN, YN, indice):
    """
    Construit l'hypercube de la séquence XN de longueur indice à laquelle
    on concatène YN de longueur indice-1
    """

    dico = {} # On initialise le dictionnaire qui va compter les occurences

    r = len(XN)# le nb de séquences (autant pour XN que pour YN)


    for real in range(r):
        # On concatene XN^i avec YN^(i-1)
```

```
        concat = tuple(XN[real][:indice] + YN[real][:indice-1])
        print(concat)

        # On l'ajoute dans le dico
        if concat not in dico.keys():
            dico[concat] = 1
        else:
            dico[concat] += 1

    for k in dico.keys():
        dico[k] = dico[k] / r

    return dico

def IM_conditionnelle(XN, YN, indice):
    """ Calcule l'entropie conditionnelle des deux séquences pour un indice précis"""

    XNYNI_1 = construire_hypercube_concat(XN, YN, indice)
    YNI = construire_hypercube(YN, indice)
    XNYNI = construire_hypercube_concat2(XN, YN, indice)
    YNI_1 = reduire(YNI)

    H_YNI_1 = entropie_via_hypercube(YNI_1)

    resultat = entropie_via_hypercube(XNYNI_1) - H_YNI_1
    + entropie_via_hypercube(YNI) - entropie_via_hypercube(XNYNI)

    return resultat


def IM_dirigee(OBSXN, OBSYN):
    """
    Calcule l'information mutuelle dirigée des deux séquences XN et YN
    """

    # On suppose que les deux séquences sont de meme taille
    if len(OBSXN[0]) != len(OBSYN[0]):
        print("Erreur, les séquences doivent être de même longueur")
        exit(1)

    n = len(OBSXN[0])
    somme = 0

    for i in range(n):
        somme += IM_conditionnelle(OBSXN, OBSYN, i)
```

```
    return somme
```

Après avoir tout testé, je me retrouve avec des résultats incohérents. Notamment $IM(X^N;Y^N) = IM(X^N \to Y^N) + IM(Y^N \to X^N)$. Je vais alors reprendre pas à pas chacune des fonctions pour les corriger.