Entropie et codage de source

Alice Andrès, Quentin SOubeyran

30 juin 2017

1 Entropie d'une distribution de probabilité

1.1 Cadre de travail et idée intuitive

Question 1

1.2 Entropie relative et information mutuelle

Question 2

$$\mathcal{D}(\mathcal{B}(a)||\mathcal{B}(b)) = a\log_2(\frac{a}{b}) + (1-a)\log_2(\frac{1-a}{1-b})$$

D'où

$$\mathcal{D}(\mathcal{B}(a)||\mathcal{B}(b)) - \mathcal{D}(\mathcal{B}(b)||\mathcal{B}(a)) = (a+b)\log_2\frac{a}{b} + (2-a-b)\log_2\frac{1-a}{1-b}$$

Or pour $a = \frac{1}{4}$ et $b = \frac{1}{2}$,

$$\mathcal{D}(\mathcal{B}(a)||\mathcal{B}(b)) - \mathcal{D}(\mathcal{B}(b)||\mathcal{B}(a)) = \frac{3}{4}\log_2(2) + \frac{5}{4}\log_2(\frac{3}{2}) \neq 0$$

Ainsi, dans le cas général, $\mathcal{D}(p||q) \neq \mathcal{D}(q||p)$

Question 3a

La fonction $-\log_2$ est strictement convexe. Alors, d'après l'inégalité de Jensen,

$$\begin{aligned}
\sum_{x\in E} \mathrm{p}(x)\left(-\log_2 \frac{q(x)}{p(x)}\right) &\geq -\log_2 \sum_{x\in E} p(x)\frac{q(x)}{p(x)} \\
&\geq -\log_2 \sum_{x\in E} q(x) \\
&\geq 0
\end{aligned}$$

Ainsi $D(p||q) \geq 0$. La stricte convexité de $-\log_2$ permet de conclure qu'il y a égalité si et seulement si $\forall x \in E$, $p(x) = q(x)$, soit $p = q$.

Question 3b

D'après Q3a, $\mathcal{I}(X,Y) = \mathcal{D}(p_{(X,Y)}||p_X \otimes p_Y) \geq 0$

Avec égalité si et seulement si $p_{(X,Y)} = p_X \otimes p_Y$ soit X et Y indépendantes.

Question 4a

$$\begin{aligned}
\mathcal{H}(X,Y) &= -\sum_{x,y\in E^2} p_{X,Y}(x,y)\log_2\left(p_{X,Y}(x,y)\right) \\
&= -\sum_{x\in E}\sum_{y\in E} p_X(x)p_{Y|X=x}(y)\log_2 p_X(x) + \log_2 p_{Y|X=x}(y) \\
&= \mathcal{H}(X) + \sum_{x\in E} p_X(x)\left(-\sum_{Y\in E} p_{Y|X=x}(y)\log_2 p_{Y|X=x}(y)\right)
\end{aligned}$$

$= \mathcal{H}(X) + \mathcal{H}(Y|X)$

Et l'on a montré l'égalité.

Question 4b

$$
\begin{aligned}
\mathcal{I}(X,Y) &= \sum_{(X,Y)\in E} p_{X,Y}(x,y)\log_2 \frac{p_{X,Y}(x,y)}{p_X(x)p_Y(y)} \\
&= \sum_{(X,Y)\in E} \mathrm{p_X}(x)p_{Y|X=x}(y)\log_2\left(p_{Y|X=x}(y)\right) - \sum_{(X,Y)\in E} p_Y(y)p_{X|Y=y}(x)\log_2\left(p_Y(y)\right) \\
&= \mathcal{H}(Y) - \mathcal{H}(Y|X) \\
&= \mathcal{H}(X) - \mathcal{H}(X|Y) \quad \text{par symétrie des rôles de X et Y)} \\
&= \mathcal{H}(Y) - (\mathcal{H}(X,Y) - \mathcal{H}(X)) \quad \text{cf. (Q4a)} \\
&= \mathcal{H}(X) + \mathcal{H}(Y) - \mathcal{H}(X,Y)
\end{aligned}
$$

Question 4c

D'après Q4b, $\mathcal{H}(X,Y) = \mathcal{H}(X) - \mathcal{I}(X;Y)$ Or $\mathcal{I}(X;Y) \geq 0$
Ainsi, $\mathcal{H}(X,Y) \leq \mathcal{H}(X)$

Question 5a

On utilise l'algorithme d'inversion de la fonction de répartition pour une loi discrète.

On utilise python pour déterminer un nombre a aléatoirement suivant la loi uniforme, entre 0 et 1, et on pose Y tel que :

$$
Y = x_i \iff \sum_{j=1}^{i-1} p_j < a \leq \sum_{j=1}^{i} p_{j+1}
$$

On peut appliquer ce principe pour $X \rightsquigarrow \mathcal{B}(\frac{1}{3})$
Soit $a \sim \mathcal{U}([0;1])$. Notons aussi $x_0 = 1$ et $x_1 = 0$
Alors $\mathbb{P}(X = x_0) = \frac{2}{3} = \mathbb{P}(a < \frac{2}{3})$ et $\mathbb{P}(X = x_1) = \frac{1}{3} = \mathbb{P}(a > \frac{2}{3})$.

Question 5b

Question 6a

Question 6b

Question 6c

2 Application au codage de source

2.1 Théorème du codage de source

Question 7a

$$\begin{aligned}\mathcal{D}(p_X||q) &= \sum_{x\in E} p_X(x)log_2(\frac{p_X(x)}{\frac{1}{c}d^{-l(x)}})\\ &\geq 0\end{aligned}$$

Alors

$$\begin{aligned}\sum_{x\in E} p_X(x)log_2(p_X(x)) \geq -\sum_{x\in E} p_X(x)l(x)log_2(d) + \sum_{x\in E} p_X(x)log_2(\frac{1}{c})\\ \iff \\ -\mathcal{H}(X) \geq -log_2(d)\mathbb{E}(X) + log_2(\frac{1}{c})\\ \geq -log_2(d)\mathbb{E}(X) \quad \texttt{(car} \leq 1)\end{aligned}$$

D'où $\frac{\mathcal{H}(x)}{log_2(d)} \leq \mathbb{E}[l(X)]$

Le cas d'égalité se déduit de celui de $\mathcal{D}$, et a lieu pour $p_X = q$, soit les $p_X(x)$ sont des puissances négatives de d.

Question 7b

Soit p une loi de probabilité telle que qui s'écrit $p_X(x) = \frac{1}{c}d^{-n_x}$ avec $c = \sum_{x\in E} d^{-n_x}$.

Cas 1 : $c \leq 1$ Prenons $\forall x \in E, l_0(x) = n_x$

Cas 2 : $c > 1$ Alors soit k tel que $\frac{c}{d^k} \leq 1$

$p_X(x) = \frac{d^k}{c}d^{-n_x-k}$, avec $\sum_{x\in E} d^{-n_x-k} \leq \frac{c}{d^k} \leq 1$

Posons alors $\forall x \in E, l_0(x) = n_x + k$

Cette application vérifie l'inégalité de Kraft-McMillan, et vérifie le cas d'égalité de la question Q7a d'après les calculs précédents pour q définie à partir de la fonction l_0.

Question 7c

La fonction puissance étant bijective sur $\mathbb{R}^+$, on a :

$$\forall x \in E, \exists \alpha_x, \quad p_X(x) = d^{\alpha_x}$$

Posons c et β tels que :

$$c = \sum_{x \in E} d^{\alpha_x} = d^{\beta}$$

Alors

$$\forall x \in E, \quad p_X(x) = \frac{1}{c} d^{-(\beta - \alpha_x)}$$

On pose donc

$$l_0(x) = \beta - \alpha_x$$

D'où

$$\begin{aligned}
\mathbb{E}[\overline{l_0}(X)] &= \sum_{x \in E} \overline{l_0}(X)\mathbb{P}(X = x) \\
&< \sum_{x \in E} l_0(X)\mathbb{P}(X = x) + \sum_{x \in E} \mathbb{P}(X = x)
\end{aligned}$$

Or d'après la question Q7a, la forme de $p_X(x) = \frac{1}{c} d^{-(\beta - \alpha_x)}$ assure :

$$\frac{\mathcal{H}(X)}{\log_2(d)} = \mathbb{E}[l(X)] \quad \text{puisque } \mathcal{D}(p_X || p_X) = 0$$

On en conclut :

$$\mathbb{E}[\overline{l_0}(X)] < \frac{\mathcal{H}(X)}{\log_2(d)} + 1$$

2.2 Mise en oeuvre de l'algorithme - L'algorithme de Huffman

Question 9a

Voici le tableau des occurrences.

a	b	c	d	e	f
2	3	1	2	2	1

On choisit c et f

a	b	d	e	cf
2	3	2	2	2

On choisit e et cf

a	b	d	ecf
2	3	2	4

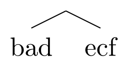
On choisit a et d

b	ad	ecf
3	4	4

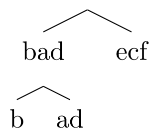
On choisit b et ad

bad	ecf
7	4

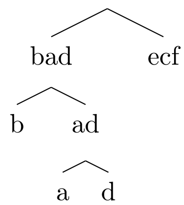
On n'a plus que deux éléments, et construisons donc l'arbre en remontant les étapes précédentes.



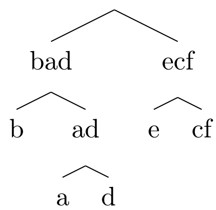
On décompose bad en b et ad



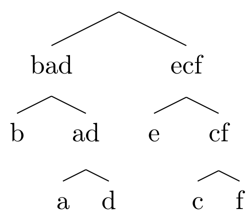
On décompose ad en a et d



On décompose ecf en e et cf



On décompose cf en c et f



On en déduit le codage de Huffman :

a	b	c	d	e	f
010	00	110	011	10	111

Question 9b

Question 9c

Question 9d

Question 9e

```
def test(a, b):
    a = b
    return b
```

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jun 20 09:30:45 2017

@author: alice
"""
import heapq
import random
import numpy as np
from CustomObjects import Token, HuffTree

import Q6


def occurencies(s):
    rep = {}
    for char in s :
        rep[char] = rep.setdefault(char, 0) + 1
    return rep
    pass

def huffman(s):
    #Creation du tas
    tas = []
    #Creation du dictionnaire des occurences
```

```
    occ = occurencies(s)
    #Initialisation du tas :
    #chaque lettre est ajoutée une fois,
    #sous la forme d’une feuille d’un arbre de Huffman
    #La classe Token redéfinie les comparaisons afin de pouvoir utiliser
    #le module heapq sans recoder les tas.
    for char in set(s) :
        heapq.heappush(tas, Token(HuffTree(char), occ[char]))

    #Fonction auxiliaire récursive
    def aux(tas):
        #extraction des des plus petits éléments
        small = heapq.heappop(tas)
        big = heapq.heappop(tas)
        if len(tas) == 0:
            #si le tas ne contenait que deux elements, on a atteint la dernière etape
            return HuffTree(small.value + big.value, (small, big))
        else :
            #s’il reste des elements, alors :
            #  - On construit l’arbre de huffman des deux plus petit elements:
            #  - On lui donne une probabilité égale à la somme des probabilités
            #    des elements qu’il represente
            #  - On continue la construction de l’arbre
            heapq.heappush(tas, Token(HuffTree(small.value + big.value,
                                               (small,big)),
                                      small._Token__comp + big._Token__comp))
            return aux(tas)
    #la fonction récursive construit un HuffTree, il n’y a plus qu’a le convertir
    #sous une forme plus utile pour l’encodage, un dictionnaire.
    return aux(tas).toDict(’’)


#questionD

def genererMot(size, chars):
#https://stackoverflow.com/questions/2257441/random-string-generation-with-upper-case-letters-a
    return ’’.join(random.choice(chars) for _ in range(size))

longueur = 26
alphabet = [’a’, ’b’, ’c’, ’d’, ’e’, ’f’, ’g’, ’h’, ’i’, ’j’, ’k’]
d = len(alphabet)
texte = genererMot(longueur, alphabet)
print(texte)
```

```
codage = huffman(texte) #attendu un dicttionnaire de la forme {char : codage}
freq = np.array([x for x in occurencies(texte).values()])
freq = freq / longueur
#entropie = Q6.Hx(freq)

#avg = np.average(np.array([x for x in codage.values()]))
#print((avg, entropie/np.log2(d) + 1))

```

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jun 20 09:30:45 2017

@author: alice
"""
import heapq
import random
import numpy as np
from CustomObjects import Token, HuffTree

import Q6


def occurencies(s):
    rep = {}
    for char in s :
        rep[char] = rep.setdefault(char, 0) + 1
    return rep
    pass

def huffman(s):
    #Creation du tas
    tas = []
    #Creation du dictionnaire des occurences
    occ = occurencies(s)
    #Initialisation du tas :
    #chaque lettre est ajoutée une fois,
    #sous la forme d'une feuille d'un arbre de Huffman
    #La classe Token redéfinie les comparaisons afin de pouvoir utiliser
    #le module heapq sans recoder les tas.
    for char in set(s) :
        heapq.heappush(tas, Token(HuffTree(char), occ[char]))

```

```
    #Fonction auxiliaire récursive
    def aux(tas):
        #extraction des des plus petits éléments
        small = heapq.heappop(tas)
        big = heapq.heappop(tas)
        if len(tas) == 0:
            #si le tas ne contenait que deux elements, on a atteint la dernière etape
            return HuffTree(small.value + big.value, (small, big))
        else :
            #s'il reste des elements, alors :
            #  - On construit l'arbre de huffman des deux plus petit elements:
            #  - On lui donne une probabilité égale à la somme des probabilités
            #    des elements qu'il represente
            #  - On continue la construction de l'arbre
            heapq.heappush(tas, Token(HuffTree(small.value + big.value,
                                               (small,big)),
                                      small._Token__comp + big._Token__comp))
            return aux(tas)
    #la fonction récursive construit un HuffTree, il n'y a plus qu'a le convertir
    #sous une forme plus utile pour l'encodage, un dictionnaire.
    return aux(tas).toDict('')


#questionD

def genererMot(size, chars):
#https://stackoverflow.com/questions/2257441/random-string-generation-with-upper-case-letters-a
    return ''.join(random.choice(chars) for _ in range(size))

longueur = 26
alphabet = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k']
d = len(alphabet)
texte = genererMot(longueur, alphabet)
print(texte)
codage = huffman(texte) #attendu un dicttionnaire de la forme {char : codage}
freq = np.array([x for x in occurencies(texte).values()])
freq = freq / longueur
#entropie = Q6.Hx(freq)

#avg = np.average(np.array([x for x in codage.values()]))
#print((avg, entropie/np.log2(d) + 1))
```