MAT 4501

Inférence bayésienne dans des modèles markoviens

Méthodes statistiques pour la segmentation dans les chaînes de Markov cachées

Eloi CAMPAGNE

Saad AMMARI

19/11/2021

Table des matières

Table des figures

Liste des tableaux

1 Segmentation supervisée d'un signal grâce aux chaînes de Markov cachées : MPM

Dans cette dernière partie du TP sur la segmentation de signal, nous étudions la méthode des chaînes de Markov cachées. Mathématiquement, cela correspond à trouver $x_s \in \Omega = \{\omega_1,\ \omega_2\}$ tel que, pour tout $s \in S$, la quantité $p(X_s = x_s \mid Y = \mathbf{y})$ soit maximale. Ici S correspond au signal étudié et l'algorithme de classification utilisé est l'algorithme du Forward-Backward.

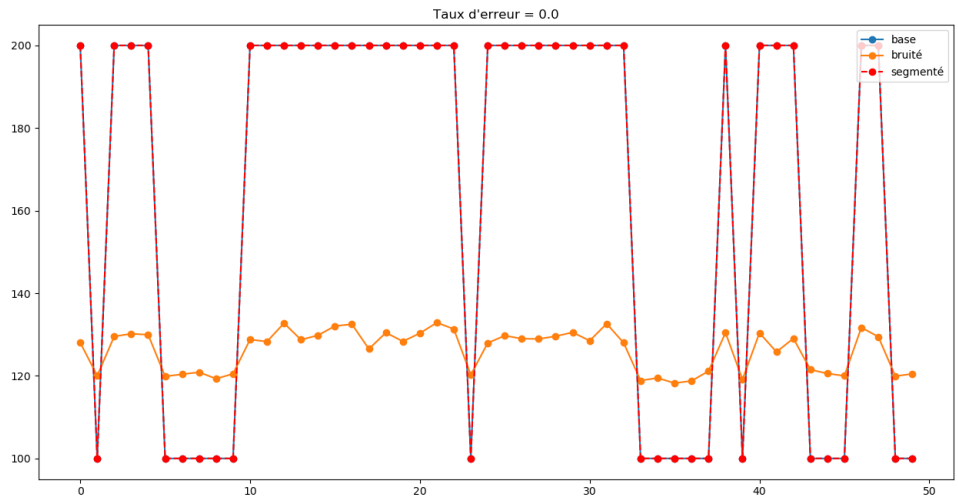

FIGURE 1.1 – Segmentation d'un signal bruité par chaîne de Markov cachée

On veut étudier l'influence des paramètres des chaînes de Markov sur la qualité de la segmentation. Pour cela, on génère plusieurs chaînes de Markov avec des para-

mètres différents (matrice de transition et probabilité à la racine), et on compare les erreurs moyennes selon que l'on segmente par la méthode MAP ou par les chaînes cachées. Pour chaque couple de paramètres, on génère une centaine de chaînes, et pour chaque chaîne, on évalue l'erreur moyenne. On en tire à la fin la moyenne des erreurs moyennes. Ces résultats sont consignés dans la Table 1.1.

Erreurs moyennes pour $T = 100$										
	Bruit 1		Bruit 2		Bruit 3		Bruit 4		Bruit 5	
	MAP	CC	MAP	CC	MAP	CC	MAP	CC	MAP	CC
Signal 1	0.00	0.00	0.20	0.13	0.29	0.23	0	0	0.40	0.32
Signal 2	0.00	0.00	0.29	0.13	0.57	0.23	0	0	0.56	0.30
Signal 3	0.00	0.00	0.51	0.03	0.51	0.10	0	0	0.51	0.16
Signal 4	0.00	0.00	0.13	0.02	0.31	0.09	0	0	0.30	0.13
Signal 5	0.00	0.00	0.18	0.18	0.31	0.31	0	0	0.38	0.38

TABLE 1.1 – Comparaison des segmentations par la méthode MAP et par les chaînes cachées

$$A_1 = \begin{pmatrix} 0.73 & 0.27 \\ 0.15 & 0.85 \end{pmatrix} \text{ et } p_{10} = 0.45 \qquad A_2 = \begin{pmatrix} 0.73 & 0.27 \\ 0.15 & 0.85 \end{pmatrix} \text{ et } p_{10} = 0.98$$

$$A_3 = \begin{pmatrix} 0.99 & 0.01 \\ 0.91 & 0.09 \end{pmatrix} \text{ et } p_{10} = 0.02 \qquad A_4 = \begin{pmatrix} 0.99 & 0.01 \\ 0.91 & 0.09 \end{pmatrix} \text{ et } p_{10} = 0.50$$

$$A_5 = \begin{pmatrix} 0.50 & 0.50 \\ 0.50 & 0.50 \end{pmatrix} \text{ et } p_{10} = 0.50$$

On remarque tout d'abord que, en moyenne, la méthode des chaînes cachées est meilleure que la méthode du maximum *a posteriori* (MAP). On peut aussi remarquer que les paramètres influent bien sur la moyenne des erreurs moyennes :

— Pour p_{10} : les signaux 1 et 2 ont la même matrice de transition, mais des probabilités à la racine différentes. De même pour les signaux 3 et 4. Pour les chaînes cachées, la moyenne des erreurs moyennes reste sensiblement la même en pas-

sant du signal 1 au signal 2, et de même en passant du signal 3 au signal 4. Ce n'est en revanche pas le cas pour la méthode du MAP, **les erreurs sont d'autant plus importantes que les probabilités à la racine sont proches de** 0 **ou de** 1**, les erreurs sont plus faibles lorsque les probabilités à la racine sont proches de** 0.50.

— Pour A : les signaux 4 et 5 ont les mêmes probabilités à la racine, mais des matrices de transition différentes. Il en va "presque" de même pour les signaux 1 et 5. Pour le MAP, la moyenne des erreurs moyennes ne varie que très peu du signal 1 au signal 5, elle varie un peu plus du signal 4 au signal 5. Cependant du signal 4 au signal 5, la moyenne des erreurs moyennes "explose" pour les chaînes cachées. Du signal 1 au signal 5, cette moyenne augmente mais relativement moins que du signal 4 au 5. On remarque que **les erreurs sont d'autant plus importantes que les probabilités de transition sont proches de** 0.50**, les erreurs sont plus faibles lorsque les probabilités de transition sont proches de** 0 **ou** 1.

Enfin, on remarque que les erreurs sont identiques lorsque les probabilités de transition suivent une loi d'équiprobabilité.

Lorsque l'on étend le script **Seg_chaines_MPM_super2.py** à la segmentation des signaux non générés par **genere_Chaine2**, on remarque que plus la longueur du signal est grande, plus l'erreur est importante. En effet, les probabilités se rapprochent de plus en plus de 0 au fur et à mesure que la taille de la chaîne augmente, à tel point qu'à partir d'un certain palier, l'ordinateur ne fait plus la différence entre une valeur très faible et 0. Les erreurs moyennes pour les différents signaux, obtenues par la méthode des chaînes cachées, sont consignées dans la Table 1.2.

Erreurs moyennes pour $T = 50$ et longueurs des signaux						
	Bruit 1	Bruit 2	Bruit 3	Bruit 4	Bruit 5	Longueur
signal.npy	0.00	0.16	0.26	0	0.33	**50**
signal1.npy	0.00	0.15	0.28	0	0.35	**250**
signal2.npy	0.50	0.50	0.50	0.50	0.50	**50000**
signal3.npy	0.50	0.50	0.50	0.50	0.50	**50000**
signal4.npy	0.50	0.50	0.50	0.50	0.50	**50000**
signal5.npy	0.50	0.50	0.50	0.50	0.50	**50000**

TABLE 1.2 – Erreurs moyennes pour les différents signaux avec l'algorithme Forward-Backward sans *rescaling* et longueurs des signaux

On remarque que pour les deux premiers signaux, leur longueur reste relativement faible par rapport aux 4 suivants. Cela se remarque dans les erreurs moyennes qui sont toutes inférieures à 0.50. Ce n'est en revanche pas le cas pour **les signaux de longueur** 50000 : pour cette longueur, la complexité du calcul effectué par l'ordinateur est trop importante et le programme n'est donc plus en mesure d'effectuer une bonne classification, **chaque classe est équiprobable**.

Pour pallier à ce problème qui n'existe pas sur le plan théorique, il est nécessaire d'implémenter une mise à l'échelle (ou *rescaling*) afin que l'ordinateur puisse effecteur des calculs avec des valeurs qui ne soient pas infinitésimales.

Erreurs moyennes															
	Bruit 1			Bruit 2			Bruit 3			Bruit 4			Bruit 5		
	MV	MAP	CC	MV	MAP	CC	MV	MAP	CC	MV	MAP	CC	MV	MAP	CC
signal.npy	0.00	0.00	0.00	0.21	0.20	0.16	0.31	0.28	0.26	0	0	0	0.43	0.36	0.34
signal1.npy	0.00	0.00	0.00	0.18	0.18	0.15	0.31	0.31	0.28	0	0	0	0.38	0.39	0.36
signal2.npy	0.00	0.00	0.00	0.18	0.18	0.16	0.31	0.31	0.29	0	0	0	0.38	0.38	0.36
signal3.npy	0.00	0.00	0.00	0.18	0.18	0.16	0.31	0.31	0.28	0	0	0	0.38	0.38	0.36
signal4.npy	0.00	0.00	0.00	0.18	0.18	0.18	0.31	0.31	0.31	0	0	0	0.38	0.38	0.38
signal5.npy	0.00	0.00	0.00	0.18	0.18	0.18	0.31	0.31	0.31	0	0	0	0.38	0.38	0.38

TABLE 1.3 – Comparaison des segmentations par la méthode du MV, du MAP et par les chaînes cachées après *rescaling*

Une fois l'étape de *rescaling* faite, l'erreur moyenne est bien moindre et ce peu importe la taille du signal. On remarque qu'en moyenne la méthode du maximum de vraisemblance (MV) est moins bonne que la méthode du maximum *a priori*, elle même moins bonne que la méthode des chaînes cachées (CC).

2 Une application à la segmentation d'images

Dans cette partie, on applique les différentes méthodes vues précédemment dans le cadre de la segmentation d'images.

On compare les taux d'erreur des différentes méthodes de segmentation pour les 5 bruits déjà définis.

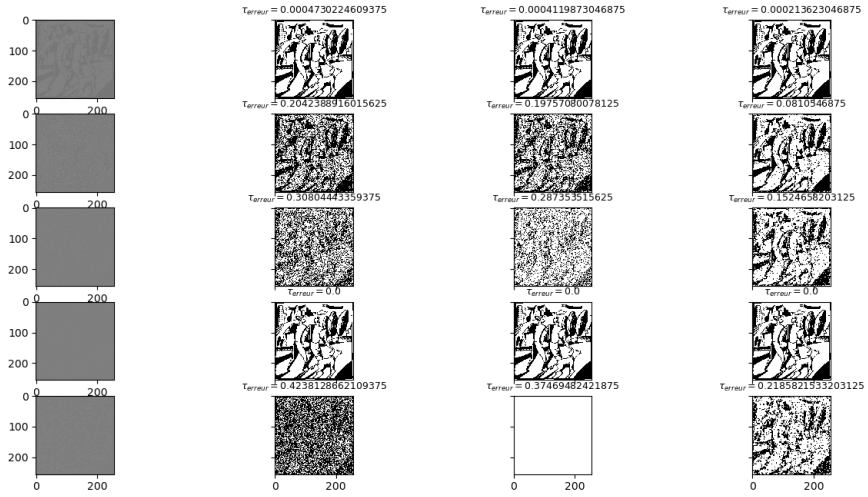

FIGURE 2.1 – Comparaison des méthodes de segmentation d'images, $p_{\text{noir}} = 0.37$ (de gauche à droite : MV, MAP, CC)

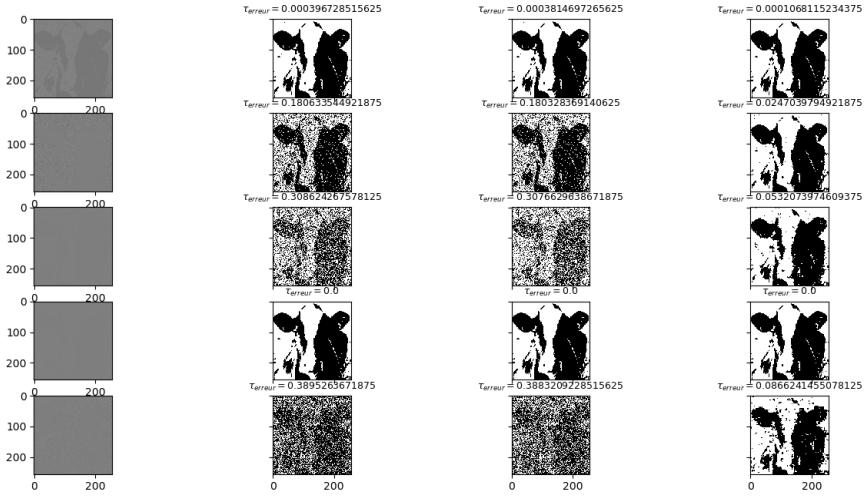

FIGURE 2.2 – Comparaison des méthodes de segmentation d'images, $p_{\text{noir}} = 0.47$ (de gauche à droite : MV, MAP, CC)

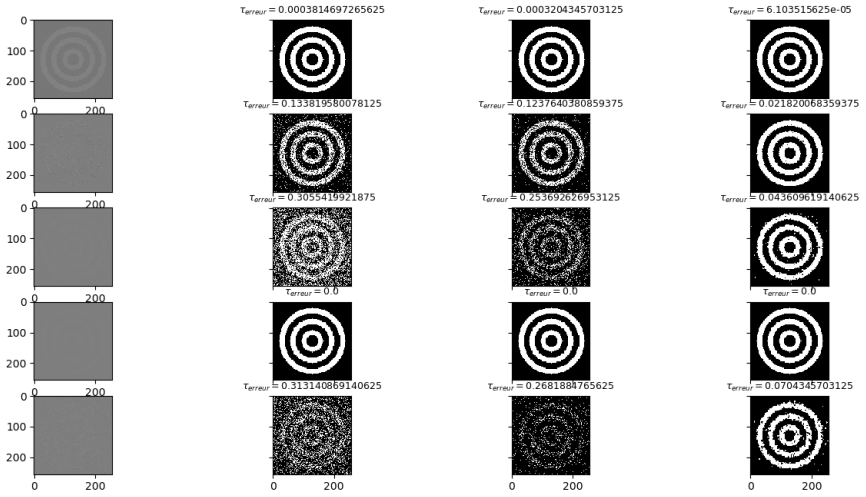

FIGURE 2.3 – Comparaison des méthodes de segmentation d'images, $p_{\text{noir}} = 0.69$ (de gauche à droite : MV, MAP, CC)

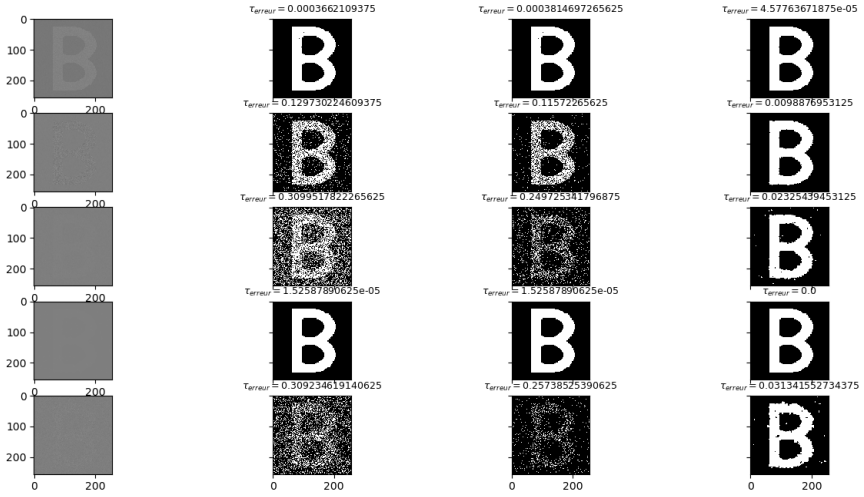

FIGURE 2.4 – Comparaison des méthodes de segmentation d'images, $p_{\text{noir}} = 0.71$ (de gauche à droite : MV, MAP, CC)

Taux d'erreur															
	Bruit 1			Bruit 2			Bruit 3			Bruit 4			Bruit 5		
	MV	MAP	CC	MV	MAP	CC	MV	MAP	CC	MV	MAP	CC	MV	MAP	CC
promenade2 ($p_{\text{noir}} = 0.37$)	0.00	0.00	0.00	0.20	0.20	0.08	0.31	0.29	0.15	0	0	0	0.42	0.37	0.22
veau2 ($p_{\text{noir}} = 0.47$)	0.00	0.00	0.00	0.18	0.18	0.02	0.31	0.31	0.05	0	0	0	0.39	0.39	0.09
cible2 ($p_{\text{noir}} = 0.69$)	0.00	0.00	0.00	0.13	0.12	0.02	0.31	0.25	0.04	0	0	0	0.31	0.27	0.07
beee2 ($p_{\text{noir}} = 0.71$)	0.00	0.00	0.00	0.13	0.12	0.01	0.31	0.25	0.02	0	0	0	0.31	0.26	0.03

TABLE 2.1 – Comparaison des méthodes de segmentation d'images

On remarque que la méthode des chaînes cachées permet une meilleure segmentation des images en moyenne. Vient ensuite la méthode du maximum *a posteriori*, et enfin la méthode du maximum de vraisemblance. Aussi, la segmentation dépend de la probabilité *a priori* (p_{noir}) : les taux d'erreur ont tendance à diminuer lorsque cette probabilité augmente, et ce peu importe le bruit (2, 3 ou 5).

3 Code Python

```
1  import numpy as np
2  import scipy.stats as scs
3  import matplotlib.pyplot as plt
4  from numpy.core.fromnumeric import argmax
5  from chain_to_image_functions import image_to_chain,
       chain_to_image
6  from skimage.io import imread
7  from PIL import Image as img
8
9  # Question 1
10
11 def gauss2(Y,br):
12     m1 = bruit(br)[0]
13     m2 = bruit(br)[1]
14     sig1 = bruit(br)[2]
15     sig2 = bruit(br)[3]
16     return np.transpose([scs.norm.pdf(Y,m1,sig1),scs.norm.pdf(Y,
           m2,sig2)])
17
18 # Question 2
19
20 def forward2_matrix(Mat_f,n,A,p10):
21     alfa = np.zeros((n,2))
```

```
22      alfa[0] = [p10, 1-p10]*Mat_f[0]
23      alfa[0] = alfa[0]/np.sum(alfa[0])
24      for t in range(1,n):
25          alfa[t] = np.dot(alfa[t-1],A)*Mat_f[t]
26          alfa[t] = alfa[t]/np.sum(alfa[t])
27      return alfa
28
29  # Question 3
30
31  def backward2_matrix(Mat_f,n,A):
32      beta = np.zeros((n,2))
33      beta[n-1] = [0.5, 0.5]
34      for t in range(n-2,-1,-1):
35          beta[t] = np.dot(beta[t+1]*Mat_f[t+1],np.transpose(A))
36          beta[t] = beta[t]/np.sum(beta[t])
37      return beta
38
39  # Question 4
40
41  def MPM_chaines2(Mat_f,n,cl1,cl2,A,p10):
42      alpha = forward2_matrix(Mat_f,n,A,p10)
43      beta = backward2_matrix(Mat_f,n,A)
44      epsilon = np.transpose(np.transpose(alpha*beta))
45      return np.where(epsilon[:,0] > epsilon[:,1], cl1, cl2)
46
47  # Question 5
48
49  def Seg_chaines_MPM_super2(n,X,cl1,cl2,br):
50      p10 = calc_probaprio2(X,cl1,cl2)[0]
51      A = calc_transit_prio2(X,n,cl1,cl2)
52      Y = bruit_gauss2(X,cl1,cl2,br)
53      Mat_f = gauss2(Y,br)
54      S = MPM_chaines2(Mat_f,n,cl1,cl2,A,p10)
55      plt.plot(X,'-o',label="base")
56      plt.plot(Y,'-o',label="bruit ")
```

```
57      plt.plot(S,'o--',c="red",label="segment ")
58      print("Matrice de transition : ",A)
59      print("Probabilit  p10 : ",p10)
60      print("Taux d'erreur : ",taux_erreur(X,S))
61      plt.title("Taux d'erreur = " + str(taux_erreur(X,S)))
62      plt.legend(loc="upper right")
63      plt.show()
64
65  # Question 7
66
67  def calc_transit_prio2(X,n,cl1,cl2):
68      a00 = 0
69      a0 = 0
70      a11 = 0
71      a1 = 0
72      for k in range(n-1):
73          if X[k] == cl1:
74              a0 += 1
75              if X[k+1] == cl1:
76                  a00 += 1
77          else:
78              a1 += 1
79              if X[k+1] == cl2:
80                  a11 += 1
81      return np.array([[a00/a0,1-a00/a0],[1-a11/a1,a11/a1]])
82
83  # Une application a la segmentation d'images
84
85  # Question 4
86
87  def Bruitage_image(X_img,br):
88      X_ch = image_to_chain(X_img)
89      cl1,cl2 = np.unique(X_ch)
90      Y = bruit_gauss2(X_ch,cl1,cl2,br)
91      return(Y,img.fromarray(chain_to_image(Y)))
```

```
92
93  def Segmentation_image_MPM(X_img,Y,br):
94      X_ch = image_to_chain(X_img)
95      n = len(X_ch)
96      cl1,cl2 = np.unique(X_ch)
97      p10 = calc_probaprio2(X_ch,cl1,cl2)[0]
98      print(p10)
99      # Y = bruit_gauss2(X_ch,cl1,cl2,br)
100     Mat_f = gauss2(Y,br)
101     A = calc_transit_prio2(X_ch,n,cl1,cl2)
102     S = MPM_chaines2(Mat_f,n,cl1,cl2,A,p10)
103     tau = taux_erreur(X_ch,S)
104     return(tau, img.fromarray(chain_to_image(S)))
105
106 def Segmentation_image_MAP(X_img,Y,br):
107     X_ch = image_to_chain(X_img)
108     cl1,cl2 = np.unique(X_ch)
109     p10 = calc_probaprio2(X_ch,cl1,cl2)[0]
110     # Y = bruit_gauss2(X_ch,cl1,cl2,br)
111     S = MAP_MPM2(Y,cl1,cl2,p10,br)
112     tau = taux_erreur(X_ch,S)
113     return(tau, img.fromarray(chain_to_image(S)))
114
115 def Segmentation_image_MV(X_img,Y,br):
116     X_ch = image_to_chain(X_img)
117     cl1,cl2 = np.unique(X_ch)
118     # Y = bruit_gauss2(X_ch,cl1,cl2,br)
119     S = classif_gauss2(Y,cl1,cl2,br)
120     tau = taux_erreur(X_ch,S)
121     return(tau, img.fromarray(chain_to_image(S)))
```