Vendredi 12 novembre 2021

Compte-rendu 2 : Méthodes statistiques pour la segmentation

dans les chaînes de Markov cachées

Alexandre Chaussard Antoine Klein

1) Apport des méthodes bayésiennes de segmentation

L'approche bayésienne de segmentation consiste à introduire une connaissance a priori sur les données. On se donne alors une fonction de perte, dans notre cas il s'agit de la perte tout ou rien, ce qui nous permet de déterminer la stratégie bayésienne de notre étude de cas.

Cette stratégie bayésienne fait alors intervenir les probabilités d'apparition des différentes classes. D'où l'introduction d'une connaissance a priori sur le système étudié.

On implémente alors en Python cette approche dite MAP (Maximum de vraisemblance a posteriori), ce qui nous donne le résultat suivant pour le signal 1 :

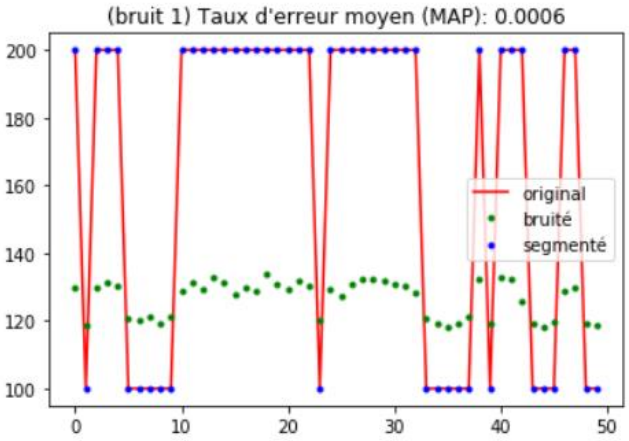

Figure 1 : Une réalisation d'un signal bruité et de sa classification par la méthode MAP

Comparons alors cette méthode MAP à la méthode du MV précédente sous les mêmes bruits de travail.

On rappelle les notations suivantes pour les bruits :

Nom du bruit	Moyenne du bruit M1	Moyenne du bruit M2	Ecart-type du bruit SIG1	Ecart-type du bruit SIG2
B1	120	130	1	2
B2	127	127	1	5
B3	127	128	1	1
B4	127	128	0,1	0,1
B5	127	128	2	3

Figure 2 : Table des différents bruits

Voici la comparaison obtenue :

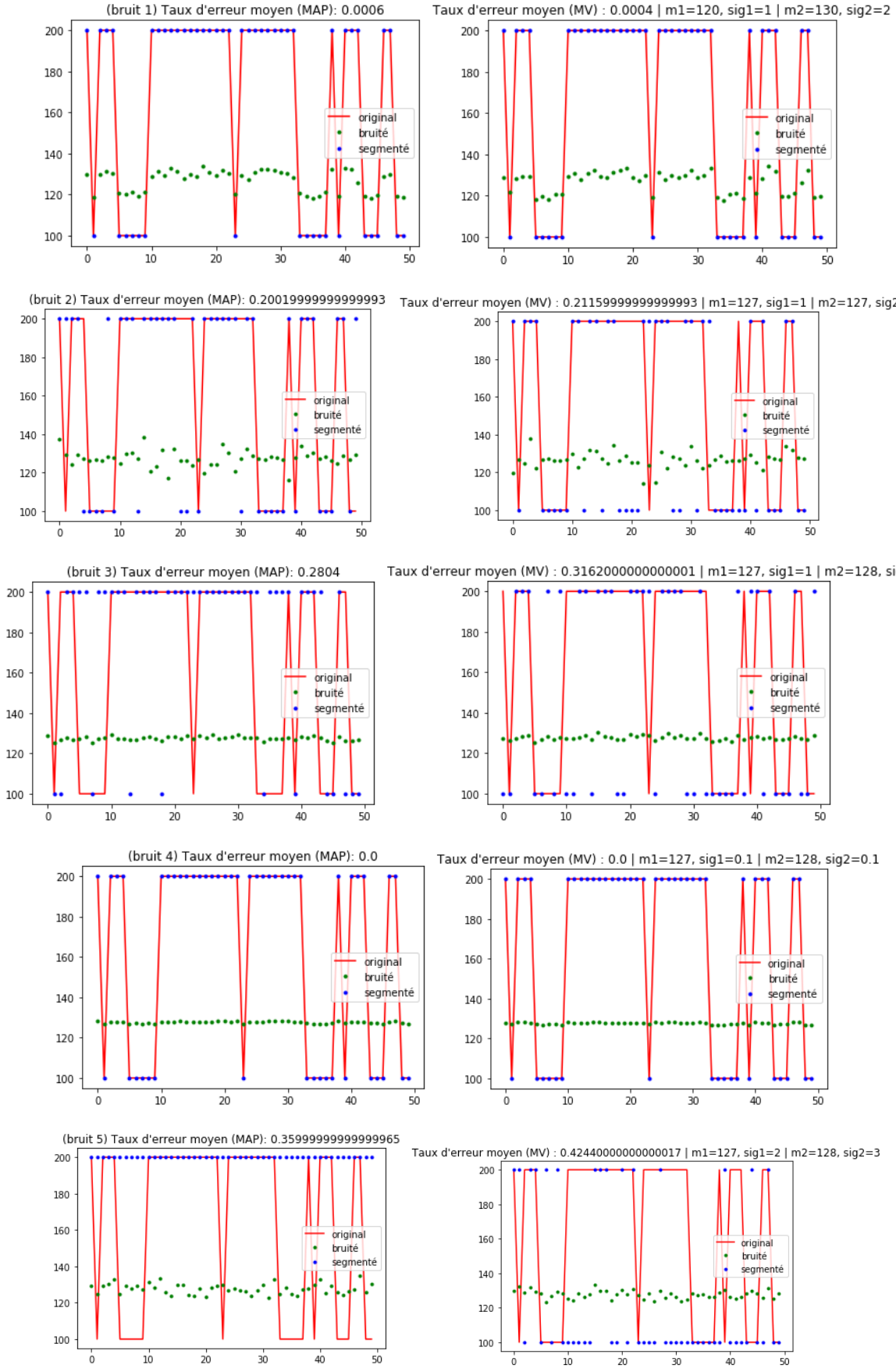

Figure 3 : Des réalisations de signaux bruités et segmentés respectivement par la méthode MAP et MV avec 200 itérations pour le calcul de l'erreur moyenne

On remarque alors que la méthode MAP donne des résultats sensiblement meilleurs que la MV. Ceci fait sens car on introduit une connaissance supplémentaire sur les classes du système pour déterminer la classe associée à chaque élément du système.

On génère ensuite des signaux avec une probabilité d'apparition des classes bien choisie selon les valeurs suivantes :

P1	0,1	0,2	0,3	0,4	0,5
P2 = 1 – P1	0,9	0,8	0,7	0,6	0,5

Figure 4 : Table des probabilités a priori d'apparition des classes

On compare alors les deux méthodes MAP et MV pour chaque bruit et chaque doublet (P1, P2), on obtient les résultats suivants :

	B1	B2	B3	B4	B5
P1 = 0,1	MV : 0% MAP : 0%	MV : 28% MAP : 8%	MV : 30% MAP : 4%	MV : 0% MAP : 0%	MV : 51% MAP : 12%
P1 = 0,2	MV : 0% MAP : 0%	MV : 26% MAP : 17%	MV : 31% MAP : 16%	MV : 0% MAP : 0%	MV : 51% MAP : 16%
P1 = 0,3	MV : 0% MAP : 0%	MV : 24% MAP : 20%	MV : 32% MAP : 31%	MV : 0% MAP : 0%	MV : 44% MAP : 34%
P1 = 0,4	MV : 0% MAP : 0%	MV : 22% MAP : 20%	MV : 32% MAP : 26%	MV : 0% MAP : 0%	MV : 42% MAP : 38%
P1 = 0,5	MV : 0% MAP : 0%	MV : 18% MAP : 19%	MV : 32% MAP : 32%	MV : 0% MAP : 0%	MV : 37% MAP : 37%

Figure 5 : Tableau récapitulatif de la segmentation des signaux bruités par les différentes classes de bruit et avec différentes probabilités d'apparition. Erreur moyenne calculée avec 200 itérations

On remarque alors que lorsque P1 = P2 = 0,5, on est dans la pire situation de classification. Cela fait sens, car on se trouve dans la situation critique de probabilité uniforme entre les deux classes, ce qui correspond bien au pire cas pour un problème de classification. A l'inverse, plus les probabilités sont éloignées, plus la MAP apparait efficace relativement à la méthode MV. En effet, la connaissance a priori sur l'apparition des différentes classes permet d'affiner la qualité du choix effectué par la méthode MAP. A l'inverse, la méthode MV n'utilise pas de connaissance a priori, de fait il ne tient pas compte de la surreprésentation d'une classe.

2) Les chaines de Markov

Les approches précédentes s'appuient sur l'hypothèse forte d'indépendance entre les différents éléments du signal. Dans cette nouvelle modélisation, nous allons supposer que le signal est un processus aléatoire dont chaque élément dépend de l'élément qui le précède uniquement, sauf la graine initiale qui ne dépend que d'une loi de génération, dans notre cas il s'agit d'une loi de Bernoulli de paramètre p = 0,5.

Le processus ainsi généré est appelé chaîne de Markov.

Voici une illustration d'un signal ainsi généré, caractérisé par sa matrice de transition également représentée.

Figure 6 : Graphe de notre chaîne de Markov avec comme loi de génération, une Bernoulli de paramètre p=0.5

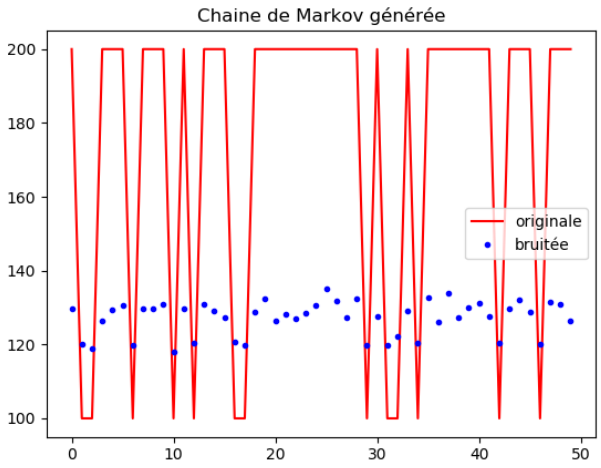

Figure 7 : Une réalisation d'un signal obtenu par chaîne de Markov puis bruité

Annexe : Nos codes Python

```python
## 1

# Calcule la loi du processus X a priori à partir du signal d'origine X
def calc_probaprio(X, cl1, cl2):
    # Paramètres de l'estimateur de p = E(X1)
    n = 0

    for x in X:
        if x == cl1:
            n += 1

    p1 = n / len(X)
    p2 = 1 - p1

    return [p1, p2]
```

```python
# Classe les éléments du signal bruité suivant le critère du MAP
def MAP_MPM2(Y, cl1, cl2, p1, p2, m1, sig1, m2, sig2):
    S = []

    for y in Y:

        if p1 * norm.pdf(y, loc=m1, scale=sig1) > p2 * norm.pdf(y, loc=m2, scale=sig2):
            S.append(cl1)
        else:
            S.append(cl2)

    return S
```

```python
## 2

# Calcul moyen du taux d'erreur
def erreur_moyenne_MAP(T, p1, p2, m1, sig1, m2, sig2, toPlot=False, forceX=np.load("signal.npy")):
    # Somme du calul de la moyenne
    tau = 0

    # Calcul de l'erreur partielle pour le plot
    erreur_partielle = []
    abscisse = []

    # Chargement des données
    X = forceX

    # Récupération des classes
    cl1, cl2 = np.unique(X)

    for k in range(0, T):
        # Bruitage du signal
        Y = bruit_gauss2(X, cl1, cl2, m1, sig1, m2, sig2)

        # Segmentation
        S = MAP_MPM2(Y, cl1, cl2, p1, p2, m1, sig1, m2, sig2)

        # Construction de la somme
        tau += taux_erreur(X, S)

        # Ajout du taux d'erreur partiel
        erreur_partielle.append(tau / (k + 1))
        abscisse.append(k + 1)
```

```python
    if toPlot:
        # Plot de l'erreur partielle
        plt.plot(abscisse, erreur_partielle)
        plt.title(
            "Taux d'erreur cumulé ((" + str(m1) + "," + str(sig1) + ") -" + " (" + str(m2) + "," + str(sig2) + "))")
        plt.show()

    return tau / T
```

```python
def test():
    M1 = [120, 127, 127, 127, 127]
    M2 = [130, 127, 128, 128, 128]
    SIG1 = [1, 1, 1, 0.1, 2]
    SIG2 = [2, 5, 1, 0.1, 3]

    # Chargement des données
    X = np.load("signal.npy")

    # Récupération des classes de X
    cl1, cl2 = np.unique(X)

    # Récupération de la loi de X à priori
    [p1, p2] = calc_probaprio(X, cl1, cl2)

    # Construction de l'axe des abscisses
    abscisse = []
    for i in range(0, len(X)):
        abscisse.append(i)

    for k in range(0, len(M1)):
        # Bruitage de l'échantillon
        Y = bruit_gauss2(X, cl1, cl2, M1[k], SIG1[k], M2[k], SIG2[k])

        # Classification
        S = MAP_MPM2(Y, cl1, cl2, p1, p2, M1[k], SIG1[k], M2[k], SIG2[k])

        # Plot

        # Plot de l'erreur partielle
        fig, ax = plt.subplots()
```

```python
        ax.plot(abscisse, X, 'r-', label="original")
        ax.plot(abscisse, Y, 'g.', label="bruité")
        ax.plot(abscisse, S, 'b.', label="segmenté")
        ax.legend()
        plt.title(
            "(bruit " + str(k + 1) + ") Taux d'erreur moyen (MAP): " + str(
                erreur_moyenne_MAP(100, p1, p2, M1[k], SIG1[k], M2[k], SIG2[k], forceX=X)))
        plt.show()
```

```python
## 3

import numpy as np


# Simule un signal de taille n composantes indépendantes à valeurs dans {cl1, cl2} avec proba {p1, p2}
def X_simu(n, cl1, cl2, p1, p2):
    X = []

    for i in range(0, n):
        u = np.random.rand()

        if u < p1:
            X.append(cl1)
        else:
            X.append(cl2)

    return X
```

```
## 4


# Comparaison des deux méthodes
def comparaison(n, cl1, cl2, P1=[0.1, 0.2, 0.3, 0.4, 0.5]):
    # Construction de l'axe des abscisses
    abscisse = []
    for i in range(0, n):
        abscisse.append(i)

    M1 = [120, 127, 127, 127, 127]
    M2 = [130, 127, 128, 128, 128]
    SIG1 = [1, 1, 1, 0.1, 2]
    SIG2 = [2, 5, 1, 0.1, 3]

    for k in range(0, len(M1)):

        for p1 in P1:
            # Déduction de p2
            p2 = 1 - p1

            # Génération du signal
            X = X_simu(n, cl1, cl2, p1, 1 - p1)
            # Bruitage
            Y = bruit_gauss2(X, cl1, cl2, M1[k], SIG1[k], M2[k], SIG2[k])

            # Segmentation selon le max de vraisemblance
            S_MV = classif_gauss2(Y, cl1, cl2, M1[k], SIG1[k], M2[k], SIG2[k])
            # Segmentation selon la méthode bayésienne
            S_MAP = MAP_MPM2(Y, cl1, cl2, p1, p2, M1[k], SIG1[k], M2[k], SIG2[k])

            # Plot
            fig, ax = plt.subplots()

            ax.plot(abscisse, X, 'r-', label="original")
            ax.plot(abscisse, Y, 'g.', label="bruité")
            ax.plot(abscisse, S_MV, 'y.', label="segmenté MV")
            ax.plot(abscisse, S_MAP, "b.", label="segmenté MAP")
            ax.legend()

            plt.title("(bruit " + str(k + 1) + " | " + str(p1) + ") Erreur moyenne : MV - " + str(
                erreur_moyenne_MV(100, M1[k], SIG1[k], M2[k], SIG2[k], forceX=X)) + " | MAP - " + str(
                erreur_moyenne_MAP(100, p1, p2, M1[k], SIG1[k], M2[k], SIG2[k], forceX=X)))

            plt.show()


# Affiche tous les graphes avec 50 points
def test2():
    comparaison(50, 100, 200)
```

```
    ##-------------- TP 3 -----------------##

## 1

# Choisit aléatoirement la classe cl1 ou clé avec les proba p1 et p2
def tirage_classe2(p1, cl1, cl2):
    u = np.random.rand()

    if u < p1:
        return cl1
    else:
        return cl2


## 2

# Génère une réalisation d'une chaîne de Markov de longueur n avec une loi initial de P10 et P20
# de matrice de transition A et de classes cl1 et cl2
def genere_Chaine2(n, cl1, cl2, A, p10):

    X = []

    u = np.random.rand()

    if u < p10:
        X.append(cl1)
    else:
        X.append(cl2)

    for i in range(1, n):

        u = np.random.rand()
```

```
        if X[i-1] == cl1:
            if u < A[0][0]:
                X.append(cl1)
            else:
                X.append(cl2)
        else:
            if u < A[1][1]:
                X.append(cl2)
            else:
                X.append(cl1)

    return X


## 3

from markovchain import MarkovChain


# Génère une réalisation de X et affiche sa représentation
def graphe_chaine2(A, plot=True):
    X = genere_Chaine2(50, 100, 200, A, 0.3)

    if plot:
        abscisse = []
        for i in range(0, len(X)):
            abscisse.append(i)

        plt.plot(abscisse, X, "r-")
        plt.show()

        mc = MarkovChain(A, ["cl1", "cl2"])
        mc.draw("./markov-chain.png")

    return X
```

```
## 4
def graphe_chaine_bruite2(A, m1, sig1, m2, sig2, cl1, cl2, plot=True):
    X = graphe_chaine2(A, True)
    Y = bruit_gauss2(X, cl1, cl2, m1, sig1, m2, sig2)

    if plot:
        abscisse = []
        for i in range(0, len(X)):
            abscisse.append(i)

        fig, ax = plt.subplots()
        ax.plot(abscisse, X, "r-", label="originale")
        ax.plot(abscisse, Y, "b.", label="bruitée")
        ax.legend()
        plt.title("Chaine de Markov générée")
        plt.show()

    return Y


def test4():
    M1 = [120, 127, 127, 127, 127]
    M2 = [130, 127, 128, 128, 128]
    SIG1 = [1, 1, 1, 0.1, 2]
    SIG2 = [2, 5, 1, 0.1, 3]

    graphe_chaine_bruite2(A=np.array([[0.2, 0.8],
                                      [0.4, 0.6]]),
                          m1=M1[0], sig1=SIG1[0],
                          m2=M2[0], sig2=SIG2[0],
                          cl1=100, cl2=200)
```