

TP : La Segmentation bayésienne

Emmanuel Monfrini, Clément Fernandes, Elie Azeraf
 (clement.fernandes@telecom-sudparis.eu,
 clementfernandes2@gmail.com)

I) Une première idée des enjeux du problème

On considère un signal ne pouvant prendre que deux valeurs distinctes ω_1 et ω_2 et on suppose que ce signal est transmis à un destinataire distant par le biais d'un canal de transmission. L'ensemble des perturbations agissant sur ce canal est modélisé par une variable aléatoire gaussienne de telle sorte que lorsque ω_1 (resp. ω_2) est transmis, on observe, au niveau du récepteur, une variable aléatoire gaussienne de moyenne m_1 (resp. m_2) et d'écart-type σ_1 (resp. σ_2).

Pour modéliser le problème on considère donc deux processus aléatoires $\mathbf{X} = (X_s)_{s \in S}$ et $\mathbf{Y} = (Y_s)_{s \in S}$. Pour tout $s \in S$, X_s prend ses valeurs dans l'espace fini des classes $\Omega = \{\omega_1, \omega_2\}$ et Y_s dans \mathbb{R} . Les réalisations de \mathbf{X} sont inobservables et le problème de la segmentation est celui de l'estimation de $\mathbf{X} = \mathbf{x} = (x_s)_{s \in S}$ à partir de l'observation $\mathbf{Y} = \mathbf{y} = (y_s)_{s \in S}$, i.e. le signal numérique à segmenter. On s'intéresse ici au cas où les couples $(X_s; Y_s)$ sont indépendants (modélisation la plus simple). En réception, la stratégie de décision \hat{s} consiste à choisir la classe ω_1 ou ω_2 pour laquelle la densité de probabilité de l'observation est la plus grande (règle du maximum de vraisemblance):

$$\hat{s}(\mathbf{y}) = \omega_i \text{ si } p(\mathbf{y}|\omega_i) = \max_{i \in \{1,2\}} p(\mathbf{y}|\omega_i)$$

0. Le langage de programmation conseillé pour ce TP est python 3 (vous pouvez en choisir un autre, mais cela peut compliquer les choses). Il vous faut alors télécharger une version de python 3, puis installer les packages suivants : « numpy », et « matplotlib » à l'aide des commandes « pip install numpy » et « pip install matplotlib ».
1. On souhaite écrire le script `premiere_idee_MV.py` réalisant les opérations précédemment décrites et calculant le taux d'erreur obtenu. Pour cela on respectera les étapes suivantes :

- a. Ecrire la fonction $Y = \text{bruit_gauss2}(X, cl1, cl2, m1, sig1, m2, sig2)$ qui bruit le vecteur X avec un bruit gaussien indépendant de moyenne $m1$ (resp. $m2$) et d'écart type $sig1$ (resp. $sig2$) pour la classe $cl1$ (resp. $cl2$). L'écriture $cl1$ (resp. $cl2$) fait référence à ω_1 (resp. ω_2) lorsque nous sommes dans le contexte de la programmation.
- b. Ecrire la fonction $S = \text{classif_gauss2}(Y, cl1, cl2, m1, sig1, m2, sig2)$ permettant de construire le signal segmenté S en classant les données du signal bruité Y dans les classes $cl1$ et $cl2$ suivant le critère précédent (on pourra utiliser la fonction `norm.pdf` qui se trouve dans `scipy.stats`).
- c. Ecrire le script `premiere_idee_MV.py` dans lequel on acquiert le signal X par `X = np.load("signal.npy")`. Le fichier `signal.npy`, ainsi que tous les autres signaux utilisés plus loin, sont téléchargeables sur `ecampus`, dans la section « Travaux Pratiques 1: La segmentation Bayésienne » du cours « IA702 - Probabilistic Models and Machine Learning ». On bruit alors le signal avec un bruit gaussien, on segmente le signal bruité suivant le critère précédent et on affiche sur un même graphique les courbes du signal original, du signal bruité et du signal segmenté.

Remarque : Pour récupérer directement les valeurs des classes dans le signal, on pourra inclure dans le script `premiere_idee_MV.py` les lignes suivantes :

```
counts, _ = np.histogram(X, bins=int(X.max()+1),
range=(0,int(X.max())) )
cl1, cl2 = np.nonzero(counts)[0]
```

2. Ecrire la fonction `tau = taux_erreur(A, B)` qui calcule et affiche le taux de signaux différents entre A et B et l'intégrer dans `premiere_idee_MV.py` pour calculer le taux d'erreur de segmentation.
3. Afin de mesurer statistiquement l'erreur, il est nécessaire de moyenner celle-ci sur un grand nombre, T , de versions bruitées du même signal, simulées à paramètres constants.
 - a. Programmer le calcul de cette erreur moyenne et tracer son évolution au cours des itérations successives
 - b. Que constate-t-on lorsque T devient très grand ?
 - c. Comment expliquer ce phénomène ?
 - d. Comment l'interpréter en termes de niveau de bruit ?

Remarque : Désormais, tout calcul d'erreur s'entend au sens de l'erreur moyenne.

4. Tester la méthode avec les 6 signaux mis à votre disposition et les bruits du tableau ci-dessous :

m1	m2	sig1	sig2
120	130	1	2
127	127	1	5

127	128	1	1
127	128	0.1	0.1
127	128	2	3

5. Présenter les résultats de segmentation dans un tableau récapitulatif et s'en servir pour déterminer ce qu'est un fort ou un faible niveau de bruit.

II) Apport des méthodes bayésiennes de segmentation

On se propose maintenant d'appliquer un autre critère de décision, plus naturel que le précédent. Il consiste à choisir la classe ω_1 ou ω_2 qui a la probabilité la plus grande d'avoir été émise, compte tenu de la valeur observée par le récepteur. C'est ce que l'on appelle le **Maximum de la vraisemblance A Posteriori** (MAP). La fonction de perte, L , du MAP (qui coïncide, dans le cas de la classification bayésienne aveugle, avec celle du MPM) est définie par :

$$L(\omega_i, \omega_j) \mapsto \begin{cases} 0 & \text{si } \omega_i = \omega_j \\ 1 & \text{sinon} \end{cases}$$

$L(\hat{s}(Y), \omega)$ désigne alors la valeur, au point (ω, \mathbf{y}) , de la fonction indicatrice du sous-ensemble de $\Omega \times \mathbf{Y}$ sur lequel \hat{s} se trompe et $E[L(\hat{s}(Y), \mathbf{X})]$ la probabilité que \hat{s} se trompe. Ainsi dans ce cas la stratégie bayésienne \hat{s}_B est définie par :

$$\hat{s}_B(\mathbf{y}) = \begin{cases} \omega_1 & \text{si } p(\omega_1|\mathbf{y}) \geq p(\omega_2|\mathbf{y}) \\ \omega_2 & \text{si } p(\omega_2|\mathbf{y}) \geq p(\omega_1|\mathbf{y}) \end{cases}$$

Notons que \hat{s}_B peut aussi s'écrire :

$$\hat{s}_B(\mathbf{y}) = \begin{cases} \omega_1 & \text{si } p(\omega_1)p(\mathbf{y}|\omega_1) \geq p(\omega_2)p(\mathbf{y}|\omega_2) \\ \omega_2 & \text{si } p(\omega_2)p(\mathbf{y}|\omega_2) \geq p(\omega_1)p(\mathbf{y}|\omega_1) \end{cases}$$

ce qui permet de faire les calculs à partir de la loi de \mathbf{X} *a priori* et des densités du bruit.

1. Programmer le script `MAP_MPM2.py` en s'inspirant du script précédent. Pour cela on écrira les fonctions :
 - a. `[p1, p2] = calc_probaprio2(X, cl1, cl2)` qui calcule la loi du processus \mathbf{X} *a priori* à partir du signal d'origine \mathbf{X} .
 - b. `S = MAP_MPM2(Y, cl1, cl2, p1, p2, m1, sig1, m2, sig2)` qui classe les éléments du signal bruité \mathbf{Y} suivant le critère du MAP (et du MPM !).
2. Tester la méthode avec les mêmes signaux et les mêmes bruits que précédemment. **Présenter les résultats dans un tableau. Comparer et commenter.**

3. Ecrire la fonction `X_simu = simul2(n,cl1,cl2,p1,p2)` qui simule un signal de taille `n` dont les composantes sont indépendantes et prennent les valeurs `cl1` et `cl2` avec les probabilités respectives `p1` et `p2`.
4. Comparer et commenter les résultats obtenus en segmentant, avec les deux méthodes et les bruits de la question 1.4, cinq signaux simulés avec `simul2` en faisant varier `p1` et `p2` (5×5 cas différents à segmenter un grand nombre de fois par chaque méthode).