

TP : Chaînes de Markov cachées et segmentation d'image

I) Segmentation bayésienne

La segmentation peut être vue comme un problème où il s'agit de rechercher des caractéristiques cachées à partir de données observables. En l'absence d'un lien déterministe entre les deux données, plusieurs segmentations possibles peuvent correspondre à une même observation. L'adoption d'une approche statistique semble donc un moyen adéquat. En effet, l'approche bayésienne de segmentation repose sur la modélisation du lien entre les deux phénomènes caché et observé.

Soit N le nombre de sites à prédire, on définit $X = (X_1, \dots, X_N)$ la variable aléatoire cachée représentant notre segmentation et $Y = (Y_1, \dots, Y_N)$ l'ensemble des données de la réalité terrain, on modélise la loi $p(x, y)$ régissant les liens entre les deux phénomènes. Le problème est alors d'estimer la réalisation de X à partir de Y . Tout estimateur de X est appelée stratégie, celle-ci est associée à une fonction de perte à minimiser.

Définissons la fonction de perte suivante pour X prenant ses valeurs dans un ensemble fini:

$$\forall (i, j) \in \Omega^2, L_{0/1}(i, j) = 1_{[i \neq j]}$$

Les deux stratégies les plus couramment utilisés dans ce cadre sont :

- L'estimateur MAP (maximum a posteriori) associé à la fonction de perte: $L_{0/1}$ sur l'ensemble des configurations de X
- L'estimateur MPM (mode des marginales a posteriori) à partir de la somme de N fonction de perte $L_{0/1}$ définies localement au niveau de chaque site n de N

Dans la suite de l'étude nous ne considérerons que la fonction de perte associée à l'estimateur MPM.

La modélisation de la loi $p(x, y)$ est au cœur du problème. Il faut trouver une modélisation (c'est-à-dire spécifier les dépendances entre les couples (X_i, Y_i) et les lois de probabilités associées) pertinente pour le cas que nous traitons. Elle doit correspondre au mieux à notre intuition ou connaissance sur le problème en question, sans être trop complexe.

Ainsi au cours de ce TP nous allons voir la modélisation par chaîne de Markov, et la comparer à la modélisation où les couples (X_i, Y_i) sont indépendants.

Pour la réalisation de ce dernier, les package python suivants sont nécessaires : « numpy » et « opencv-python ». Pour les installer, il suffit d'ouvrir une console python et de taper la commande « pip install numpy » et « pip install opencv-python ». Par ailleurs le code fournit avec ce TP fonctionne sous python 3, et n'est pas garanti de fonctionner sous python 2.

II) Les chaînes de Markov

Intuitivement, lorsque l'on classe une variable X_i d'un signal, regarder celles qui la précèdent dans le signal devrait apporter de l'information supplémentaire. Tenir compte des valeurs des autres variables revient à supposer que les différentes variables aléatoires composant le signal ne sont pas nécessairement indépendantes. L'objectif des modélisations

par chaînes de Markov est d'introduire des modèles permettant de tenir compte de cette dépendance.

Soit $X = (X_1, \dots, X_N)$ un processus caché qui prend ses valeurs à partir d'un ensemble fini de classes $\Omega = (\omega_1, \dots, \omega_k)$ et qui doit être estimé à partir d'un processus observable $Y = (Y_1, \dots, Y_N)$ qui prend ses valeurs dans \mathbb{R} . Le processus couple (X, Y) est dit chaîne de Markov cachée si X est une chaîne de Markov et si (X, Y) est de Markov.

$$\text{Soit } p(x, y) = p(x_1)p(y_1|x_1) \prod_{n=2}^N p(x_n|x_{n-1})p(y_n|x_n)$$

Nous supposons dans la suite que les probabilités de transitions ne dépendent pas de N , la chaîne est dite homogène. La loi de X est alors déterminée par la loi de X_1 et la matrice de transition A .

1. Remplir les fonctions suivantes (elles se trouvent dans le script « tools.py »):
 - `bruit_gauss(signal, w=[w1,w2], m1,sig1,m2,sig2)` qui bruit le signal avec un bruit gaussien indépendant de moyenne $m1$ (resp. $m2$) et d'écart type $sig1$ (resp. $sig2$) pour la classe $w1$ (resp. $w2$).
 - `gauss(signal_noisy, m1, sig1, m2, sig2)` qui construit la matrice de dimension $nx2$ contenant les transformations du processus bruité Y par les densités du bruit correspondant à chaque classe.
 - `calc_erreur(signal1, signal2)` qui calcule le taux d'erreur entre deux signaux.
2. Ouvrir le script « markov_chain.py » et Remplir la fonction `forward(A, p, gauss)` qui calcule récursivement les composantes de la matrice α de dimension $nx2$ par la procédure forward.
3. Remplir la fonction `backward(A, gauss)` qui calcule récursivement les composantes de la matrice β de dimension $nx2$ par la procédure backward.
4. Remplir la fonction `mpm_mc(signal_noisy, w, p, A, m1, sig1, m2, sig2)` qui segmente le signal en suivant le critère du MPM pour les chaînes de Markov cachées.
5. On donne la fonction `simu_mc(n, w=[w1,w2], p, A)` qui permet de générer des signaux de taille n selon une chaîne de Markov. Ecrire le script `markov_chain_segmentation` qui génère une réalisation d'une chaîne de Markov, l'affiche, la bruit et affiche le signal bruité, le segmente suivant le critère du MPM dans les chaînes de Markov cachées et affiche le signal segmenté ainsi que le taux d'erreur.

6. On se donne les bruits suivants :

m1	m2	sig1	sig2
0	3	1	2
1	1	1	5
0	1	1	1

Générer trois signaux de taille N avec des matrices de transitions différentes (à choisir) et segmenter ces signaux avec chacun des trois bruits du tableau. Présenter les résultats dans un tableau récapitulatif et commenter.

Rq : Pour obtenir des taux d'erreur pertinents, on a deux choix. Soit on prend N relativement petit (50) et on répète la génération et segmentation plusieurs fois pour les mêmes paramètres, on moyenne ensuite les taux d'erreur. Soit on fait une seule segmentation par paramètre avec N grand (1000).

7. Remplir la fonction `calc_probaprio_mc(signal, w)` qui estime, grâce aux estimateurs empiriques des fréquences, les probabilités de transition de la chaîne de Markov X et renvoie la matrice de transition A.
8. Dans le script « `gaussian_mixture.py` » vous trouverez les fonctions permettant de simuler (`simu_gm(n, w, p)`), segmenter par MPM (`mpm_gm(signal_noisy, w, p, m1, sig1, m2, sig2)`) et estimer les probabilité d'apparitions des classes du signal (`calc_probaprio_gm(signal, w)`) pour un modèle où les couples (X_i, Y_i) sont indépendants. Remplir le script « `compare_markov_indep.py` » en réalisant l'étude suivante:

- Générer un signal par le modèle (X_i, Y_i) indépendants
- Bruiter le signal
- Utiliser la fonction `calc_probaprio_mc(signal, w)` sur le signal non bruité pour obtenir les paramètres d'une chaîne de Markov
- Restaurer le signal par MPM suivant le modèle indépendant et par MPM suivant la chaîne de Markov. Comparer le taux d'erreur pour les deux segmentations.
- Générer un signal par une chaîne de Markov
- Bruiter le signal
- Utiliser la fonction `calc_probaprio_gm(signal, w)` sur le signal non bruité pour obtenir les paramètres du modèle indépendant.
- Restaurer le signal par MPM suivant le modèle indépendant et par MPM suivant la chaîne de Markov. Comparer le taux d'erreur pour les deux segmentations.

Ici même remarque que précédemment, soit on génère des « petit signaux » et on répète l'opération plusieurs fois pour les mêmes paramètres, et on moyenne les taux d'erreur. Soit on le fait une fois par signal, avec N grand

On fera cette étude pour tous les bruits du tableau. On pourra prendre par exemple comme paramètre pour le modèle indépendant $p = (0.25, 0.75)$ et pour la chaîne de Markov $p = (0.25, 0.75)$ et $A = \begin{bmatrix} 0.8 & 0.2 \\ 0.07 & 0.93 \end{bmatrix}$

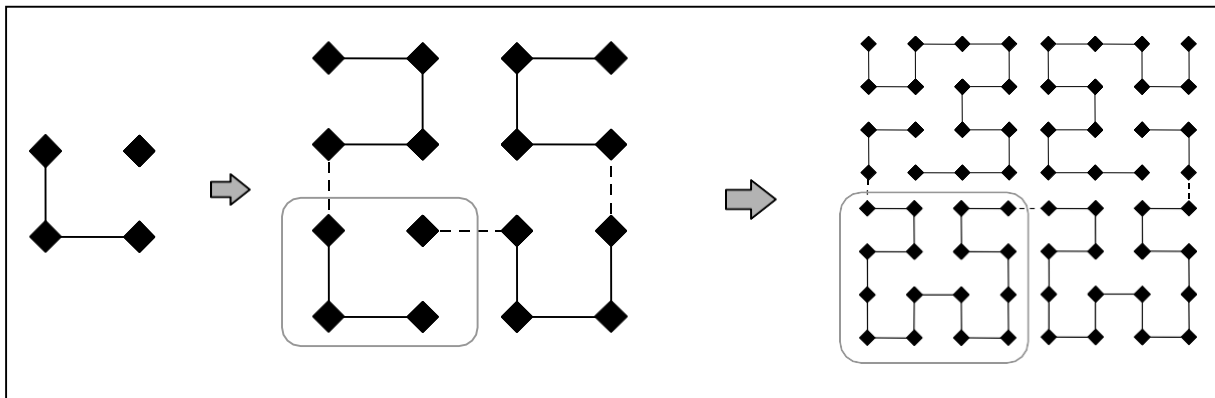
Présenter les résultats et commenter.

III) Application à la segmentation d'images

Bien qu'une chaîne de Markov soit, par nature, un processus stochastique permettant de décrire des phénomènes à une dimension, il est possible d'utiliser cette modélisation dans le cadre de problématiques naturellement décrites en 2 dimensions.

Si l'on considère l'exemple de la segmentation d'images (satellites, radar, infrarouge etc...), il est naturel de penser que les interactions entre les pixels voisins se décrivent favorablement en 2 dimensions. Pourtant, rien n'interdit de parcourir l'image, ligne par ligne, colonne par colonne ou même autrement, et de considérer que la suite de pixels ainsi constituée peut être modélisée par un chaîne de Markov.

L'expérience a montré que la meilleure façon de parcourir était de suivre un parcours de Peano :



On va se placer dans un contexte un peu différent que les autres parties, l'idée ici est d'être capable de segmenter une image bruitée sans avoir l'image original ni les paramètres des modèles. On va donc devoir estimer les paramètres de manière non-supervisée. Cela peut être fait dans les deux modèles grâce à l'algorithme EM.

1. Dans le script « markov_chain.py », remplir la fonction `calc_param_EM_mc(signal_noisy, p, A, m1, sig1, m2, sig2)` qui donne l'estimation des nouveaux paramètres par EM lors d'une itération. Cette fonction sera appelée par la fonction `estim_param_EM_mc(iter, signal_noisy, p, A, m1, sig1, m2, sig2)` déjà codée, qui réalise l'algorithme EM complet.
2. Dans le dossier « images » se trouve des images noir et blanc. Dans le script « image_segmentation.py », réaliser l'étude suivante :
 - Choisir une image, l'aplatir avec la fonction `peano_transform_img(img)`
 - Bruiter le signal 1D ainsi obtenu

- Utiliser la fonction `estim_param_EM_mc(iter, signal_noisy, p, A, m1, sig1, m2, sig2)` sur le signal bruité pour obtenir les paramètres d'une chaîne de Markov
- Restaurer le signal par MPM suivant la chaîne de Markov
- Utiliser la fonction `estim_param_EM_gm(iter, signal_noisy, p, A, m1, sig1, m2, sig2)` pour obtenir les paramètres du modèle indépendant
- Restaurer le signal par MPM suivant le modèle indépendant.

On fera cette étude pour 3 images choisies dans le dossier, avec les 3 bruits évoqués précédemment. Présenter les résultats et commenter.