TP : La Segmentation bayésienne d'image

Emmanuel Monfrini, Clément Fernandes, Elie Azeraf

Contact : clement.fernandes@telecom-sudparis.eu

Ce TP est à rendre pour le 10/05/2021, à 23h59.

I) La segmentation bayésienne d'image

Le but de ce TP est de se familiariser avec un ensemble de méthodes permettant d'opérer une classification de données cachées (clustering de données). Cette problématique, très générale, apparaît dans tous les domaines pour lesquels les variables d'intérêt ne sont pas directement observables, que ce soit partiellement ou dans leur totalité.

Pour illustrer cela, on considère le problème suivant: nous avons reçu une image à deux classes (noir et blanc) bruitée à cause du processus de transmission et nous voulons retrouver notre image d'origine. Pour modéliser le problème on considère donc deux processus aléatoires $\boldsymbol{X} = (X_s)_{\forall s \in S}$ et $\boldsymbol{Y} = (Y_s)_{\forall s \in S}$. 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 R. **Les réalisations de $\boldsymbol{X}$ sont considérées inobservables dans toute la suite du TP**, et le problème de la segmentation est celui de l'estimation de $\boldsymbol{X} = \boldsymbol{x} = (x_s)_{\forall s \in S}$ à partir de l'observation $\boldsymbol{Y} = \boldsymbol{y} = (y_s)_{\forall s \in S}$, i.e. notre image bruitée.

Notre approche pour la résolution de ce problème va suivre la modélisation probabiliste, ainsi il nous faut d'abord une hypothèse sur les dépendances des variables aléatoires (il nous faut établir notre graphe de probabilité). Dans la suite du TP nous allons considérer deux hypothèses :

- L'hypothèse où les couples (X_s, Y_s) sont indépendants (C'est le modèle de mélange de gaussiennes).
- L'hypothèse où $(\boldsymbol{X}, \boldsymbol{Y})$ forme une chaîne de Markov cachée stationnaire.

Il nous faut ensuite, pour chaque graphe de probabilité considéré, faire une hypothèse sur la forme des lois. Nous allons voir ces hypothèses dans les parties de ce TP dédiées à chacun des modèles

Puis nous allons devoir estimer nos paramètres en fonction des données observables. Ici nous allons donc estimer les paramètres pour chaque graphe à l'aide des observations $\boldsymbol{y} = (y_s)_{\forall s \in S}$ seulement.

Enfin, une fois que nous aurons les lois complètes pour les deux modèles, il faudra choisir une fonction d'erreur et une stratégie de prédiction minimisant celle-ci. Dans notre cas, le choix de la fonction d'erreur est plutôt simple. En effet, il n'y a pas vraiment d'erreurs « plus grave » que d'autres (le but étant seulement de retrouver notre image noir et blanc d'origine). On a donc deux choix :

- $L(\boldsymbol{x}^{réel}, \widehat{\boldsymbol{x}}) = \sum_{s \in S} 1_{[x_s^{réel} \neq \hat{x}_s]}$ qui correspond à la stratégie bayésienne du MPM
- $L(\boldsymbol{x}^{réel}, \widehat{\boldsymbol{x}}) = \begin{cases} 0 \; si \; \boldsymbol{x}^{réel} = \widehat{\boldsymbol{x}} \\ 1 \; si \; \boldsymbol{x}^{réel} \neq \widehat{\boldsymbol{x}} \end{cases}$ qui correspond à la stratégie bayésienne du MAP

Dans la suite nous allons choisir la fonction d'erreur $L(\boldsymbol{x}^{réel}, \widehat{\boldsymbol{x}}) = \sum_{s \in S} 1_{[x_s^{réel} \neq \hat{x}_s]}$ et donc la stratégie du MPM. On peut noter que dans le cas de l'hypothèse des couples indépendants, MAP et MPM sont confondus.

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

1. Le répertoire `images` contient les images utilisées dans ce TP et le répertoire `codes` contient le script `utils.py` où sont déjà codées certaines des fonctions nécessaires à ce TP.

II) Modèle des couples indépendants

On se place dans le cas où les couples (X_s, Y_s) sont indépendants et de même loi. La loi de $(\boldsymbol{X}, \boldsymbol{Y})$ s'écrit de cette façon **:**

$$p(\boldsymbol{x}, \boldsymbol{y}) = \prod_{s \in S} p(x_s) p(y_s | \, x_s)$$

Il nous faut maintenant préciser la forme des lois. Au vu du problème, c'est facile. X_s sera une variable aléatoire discrète, à deux classes. Caractérisée par la loi $p(X_s = \omega_1) = \pi$ et $p(X_s = \omega_2) = 1 - \pi$. Les lois de Y_s conditionnelles a $X_s = \omega_1$ et $X_s = \omega_2$ seront gaussiennes, respectivement f_1 paramétrées par μ_1, σ_1 et f_2 paramétrées par μ_2, σ_2

1. Ecrire la fonction `bruit_gauss2(X,cl1,cl2,m1,sig1,m2,sig2)` qui bruite 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.

2. Ecrire la fonction `taux_erreur(A,B)` qui calcule et affiche le taux de signaux différents entre A et B.
3. Ecrire la fonction `calc_probaprio2(X,cl1,cl2)` qui calcule la loi du processus X a priori à partir du signal d'origine X.
4. On souhaite écrire le script `Segmentation_image_indep.py` qui acquiert, bruite, estime les paramètres sur la version bruitée puis segmente une image (carré dont la longueur du côté est une puissance de 2) à deux classes selon le modèle indépendant. On fournit pour cela les fonctions `MPM_gm(Y, cl1, cl2, p1, p2, m1, sig1, m2, sig2)` et `estim_param_EM_gm(iter, Y, p1, p2, m1, sig1, m2, sig2)` (elles se trouvent dans le script `utils.py`). Nous allons procéder aux étapes suivantes :
    a. Charger l'image avec la fonction `cv.cvtColor(cv.imread('path_to_image'),cv.COLOR_BGR2GRAY)`
    b. Transformer l'image 2 dimensions en tableau une dimension grâce à la fonction `line_transform_img(img)` du script `utils.py`
    c. Bruiter l'image avec la fonction `bruit_gauss2`
    d. Initialiser les paramètres de l'algorithme EM en segmentant rapidement l'image bruitée Y par un kmeans (package `scikit-learn`), puis en calculant les probabilités a priori avec `calc_probaprio2`, les moyennes et les variances grâce au résultat de la segmentation par le kmeans.
    e. Utiliser la fonction `estim_param_EM_gm` pour estimer les paramètres du modèle par l'algorithme EM.
    f. Utiliser la fonction `MPM_gm` pour segmenter l'image
    g. Calculer le taux d'erreur entre l'image segmentée et l'image réelle
    h. Afficher l'image bruitée, l'image segmentée et l'image réelle. Pour retransformer votre signal une dimension en image, on pourra utiliser la fonction `transform_line_in_img(signal, dSize)` du script `utils.py` et pour afficher une image on pourra utiliser `cv.imshow(title, img)`.

5. Tester la méthode pour 3 images bien choisies dans le dossier, avec les 3 bruits du tableau ci-dessous. Présenter les résultats (taux d'erreur et images segmentées) et commenter. Pour le choix des images, choisir une image parmi (alpha2, beee2 et cible2), une image parmi (country2, promenade2 et veau2) et une image parmi (zebre2 et city2).

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

III) Modèle de chaîne de Markov cachées

On se place dans le cas où $(\boldsymbol{X}, \boldsymbol{Y})$ forme une chaîne de Markov cachée stationnaire. La loi de $(\boldsymbol{X}, \boldsymbol{Y})$ s'écrit de cette façon **:**

$$p(\boldsymbol{x}, \boldsymbol{y}) = p(x_1)p(y_1|x_1)\prod_{n=1}^{N-1} p(x_{n+1}|x_n)p(y_{n+1}|x_{n+1})$$

Il nous faut maintenant préciser la forme des lois. Au vu du problème, c'est facile. Les X_n seront des variables aléatoires discrètes, à deux classes, caractérisées par la loi $p(X_0 = \omega_1) = \pi_1$ et $p(X_0 = \omega_2) = \pi_2$. Et les lois conditionnelles $p\big(X_{n+1} = \omega_j | X_n = \omega_i\big) = a_{i,j}$. Les lois de Y_n conditionnelles a $X_n = \omega_1$ et $X_n = \omega_2$ seront gaussiennes, respectivement f_1 paramétrées par μ_1, σ_1 et f_2 paramétrées par μ_2, σ_2.

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 :

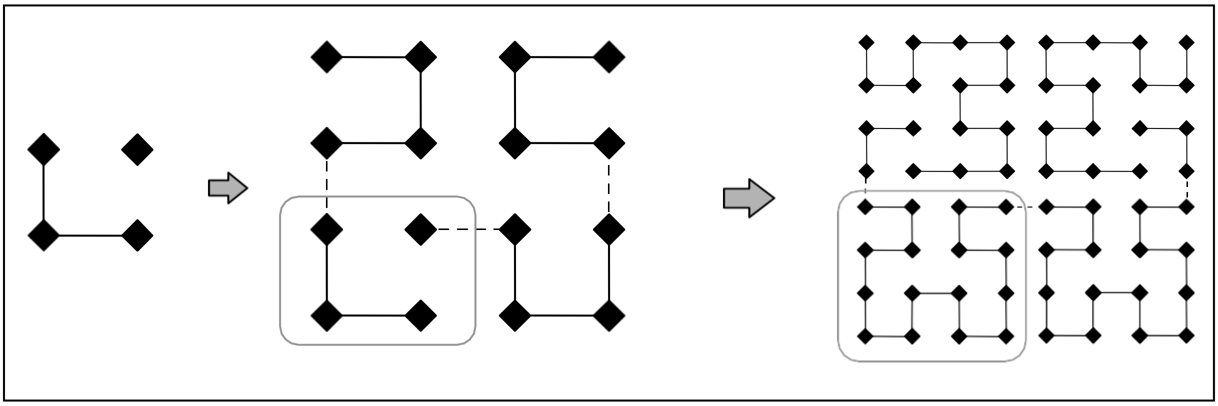

Figure 1: Parcours de peano d'un carré

1. Commencer par construire une matrice `Mat_f` destinée à simplifier les calculs des probabilitées forward et backward. Ecrire pour cela la fonction `Mat_f = gauss2(Y,n,m1,sig1,m2,sig2)` qui construit la matrice de dimension

n*2 contenant les transformations du processus bruité Y par les densités du bruit correspondant à chaque classe.

2. Ecrire la fonction `forward2(Mat_f,A,p10,p20)` qui calcule récursivement les composantes de la matrice `alfa` de dimension n*2 par la procédure forward. A est la matrice de transition. **Attention, ici on travaille avec des séquences longues. Il faudra donc penser à utiliser la version de l'algorithme avec le rescaling.**
3. Ecrire la fonction `backward2(Mat_f,A)` qui calcule récursivement les composantes de la matrice `beta` de dimension nx2 par la procédure backward. **Attention, ici on travaille avec des séquences longues. Il faudra donc penser à utiliser la version de l'algorithme avec le rescaling.**
4. Ecrire la fonction `calc_probatrans2(X,cl1,cl2)` qui calcule la matrice de transition du processus X a priori à partir du signal d'origine X.
5. Ecrire la fonction `MPM_chaines2(Mat_f,n,cl1,cl2,A,p10,p20)` qui segmente le signal en suivant le critère du MPM pour les chaînes de Markov cachées.
6. En vous inspirant de la fonction `estim_param_EM_gm`, écrire la fonction `estim_param_EM_mc(iter, Y, A, p10, p20, m1, sig1, m2, sig2)`, qui estime les paramètres `A, p10, p20, m1, sig1, m2, sig2` par l'algorithme EM, à partir des observations `Y`.
7. On souhaite écrire le script `Segmentation_image_mc.py` qui acquiert, bruite, estime les paramètres sur la version bruitée puis segmente une image (carré dont la longueur du côté est une puissance de 2) à deux classes selon le modèle de chaîne de Markov cachées. Nous allons procéder aux étapes suivantes :
   a. Charger l'image avec la fonction `cv.cvtColor(cv.imread('path_to_image'),cv.COLOR_BGR2GRAY)`
   b. Transformer l'image 2 dimensions en tableau une dimension grâce à la fonction `peano_transform_img(img)` du script `utils.py`
   c. Bruiter l'image avec la fonction `bruit_gauss2`
   d. Initialiser les paramètres de l'algorithme EM en segmentant rapidement l'image bruitée Y par un kmeans (package `scikit-learn`), puis en calculant les probabilités a priori avec `calc_probaprio2`, les transitions a priori avec `calc_probatrans2`, les moyennes et les variances grâce au résultat de la segmentation par le kmeans.
   e. Utiliser la fonction `estim_param_EM_gm` pour estimer les paramètres du modèle par l'algorithme EM.
   f. Utiliser la fonction `MPM_gm` pour segmenter l'image
   g. Calculer le taux d'erreur entre l'image segmentée et l'image réelle
   h. Afficher l'image bruitée, l'image segmentée et l'image réelle. Pour retransformer votre signal une dimension en image, on pourra utiliser la fonction `transform_peano_in_img(signal, dSize)` du script

`utils.py` et pour afficher une image on pourra utiliser `cv.imshow(title, img)`.

8. Tester la méthode pour les 3 images choisies précédemment, avec les 3 bruits précédemment utilisés. Présenter les résultats (taux d'erreur et images segmentées) et commenter, en comparant les résultats avec ceux du modèle indépendant.