TME 8 : Algorithme k-means, Inpainting

K-means

Le principe de l'algorithme k-means est de trouver une partition de l'espace d'entrée en considérant la densité d'exemples pour caractériser ces partitions. Un cluster C_i correspond à la donnée d'un prototype $\mu_i \in \mathbb{R}^d$ dans l'espace d'entrée. Chaque exemple x est affecté au cluster le plus proche selon une distance (euclidienne ou autre) entre l'exemple et le prototype du cluster. Soit $s_C : \mathbb{R} \to \mathbb{N}$ la fonction d'affectation associé au clustering $C = \{C_1, C_2, \ldots, C_k\}$: $s_C(x) = argmin_i \|\mu_i - x\|^2$. La fonction de coût sur un ensemble de données $\{x_1, \ldots, x_n\}$ généralement considérée dans ce cadre est la moyenne des distances intra-clusters : $\frac{1}{n}\sum_{i=1}^{n}\sum_{j|s_C(x_j)=i}\|\mu_i - x_j\|^2 = \frac{1}{n}\sum_{i=1}^{n}\|\mu_{s_C(x_i)} - x_i\|^2$. C'est également ce qu'on appelle le coût de reconstruction : effectivement, dans le cadre de cette approche, chaque donnée d'entrée peut être "représentée" par le prototype associé : on réalise ainsi une compression de l'information (n.b. : beaucoup de liens existent entre l'apprentissage et la théorie de l'information, la compression et le traitement de signal). L'algorithme fonctionne en deux étapes, (la généralisation de cet algorithme est appelé algorithme E-M, Expectation-Maximization) :

- à partir d'un clustering C^t, les prototypes $\mu_i^t = \frac{1}{|C_i|}\sum_{x_j \in C_i^t} x_j$, barycentres des exemples affectés à ce cluster ;
- à partir de ces nouveaux barycentres, calculer la nouvelle affectation (le prototype le plus proche).

Ces deux étapes sont alternées jusqu'à stabilisation.

Cet algorithme peut être utilisé pour faire de la compression d'image (connu également sous le nom de quantification vectorielle). Une couleur est codée par un triplet (r, g, b) qui dénote le mélange de composantes rouge, vert et bleu. En limitant le nombre de couleurs possibles dans l'image, on réalise une compression en limitant la longueur de codage de chaque couleur. L'objectif est de trouver quelles couleurs doivent être présentent dans notre *dictionnaire* afin de minimiser l'erreur entre l'image compressée et l'image originale (remarque : c'est exactement l'erreur de reconstruction ci-dessus). En considérant l'ensemble des pixels de l'image comme la base d'exemples non supervisée, le nouveau codage de chaque pixel peut être obtenu par le résultat de l'algorithme k-means sur cette base d'exemple.

Le bout de code suivant permet de lire, afficher, modifier, sauver une image au format `png` et de la stocker dans un tableau de taille $l \times h \times c$, l la largeur de l'image, h la hauteur, et c 3 généralement pour les 3 couleurs (parfois 4, la 4eme dimension étant pour la transparence, non utilisée ici).

```
import matplotlib.pyplot as plt

im=plt.imread("fichier.png")[:,:,:3] #on garde que les 3 premieres composantes, la tra
im_h,im_l,_=im.shape
pixels=im.reshape((im_h*im_l,3)) #transformation en matrice n*3, n nombre de pixels
imnew=pixels.reshape((im_h,im_l,3)) #transformation inverse
plt.imshow(im) #afficher l'image
```

Implémentez l'algorithme k-means. Expérimentez la compression : choisissez une image, construisez avec k-means l'image compressée et affichez là. Etudier en fonction du nombre de clusters (couleurs) choisis comment évolue l'erreur de reconstruction.

Quel est le gain en compresssion effectué ?

Sachant que souvent une image peut être découpée en région de tonalité homogène, voyez-vous une amélioration possible pour augmenter la compression tout en diminuant l'erreur de compression ?

Inpainting

L'*inpainting* en image s'attache à la reconstruction d'images détériorées ou au remplissage de parties manquantes (éliminer une personne ou un objet d'une image par exemple). Cette partie est consacrée à l'implémentation d'une technique d'inpainting présentée dans [1] utilisant le Lasso et sa capacité à trouver une solution parcimonieuse en termes de poids. Ces travaux sont inspirés des recherches en *Apprentissage de dictionnaire et représentation parcimonieuse* pour les signaux [2]. L'idée principale est de considérer qu'un signal complexe peut être décomposé comme une somme pondérée de signaux élémentaires, comme c'est le cas également dans l'ACP. les signaux élémentaires - nommés atomes - peuvent être cependant dans ce cadre fortement redondants. La difficulté est donc double : réussir à construire un dictionnaire d'atomes - les signaux élémentaires - et trouver un algorithme de décomposition/recomposition d'un signal à partir des atomes du dictionnaire, i.e. trouver des poids parcimonieux sur chaque atome tels que la combinaison linéaire des atomes permettent d'approcher le signal d'origine. Dans la suite, nous étudions une solution pour la deuxième question à l'aide de l'algorithme du Lasso.

Lasso Le Lasso est un modèle linéaire qui s'inspire de la régression linéaire où la norme de pénalisation l_2 au carré est remplacée par une norme l_1. On résout dans ce cas le problème suivant :

$$\hat{f} = argmin_w \frac{1}{2n} \sum_{i=1}^{n} \|Xw - Y\|_2^2 + \lambda \cdot \|w\|_1$$

L'algorithme est implémenté dans scikit-learn dans le module `sklearn.linear_model.Lasso`. Plus de détails sur l'implémentation sont disponibles dans la documentation de scikit-learn.

Formalisation Pour une image de dimension 2, nous noterons $p_i \in [0,1]^3$ le i-ème pixel de l'image exprimée sur 3 canaux [1]. Un patch de taille h centré au pixel p_i correspond a un petit carré de pixels dans l'image de longueur h donc le centre est le pixel p_i. Nous le noterons par la suite Ψ^{p_i} (la taille h sera constante). Ce patch correspond à un tenseur (matrice 3d ici) de taille $h \times h \times 3$ que l'on peut voir sans perte de généralité comme un vecteur colonne de taille $3h^2$; à une image de taille $(width, height)$, il est possible d'associer l'ensemble des patchs $\mathbf{\Psi}$. L'hypothèse fondamentale est qu'une image a une cohérence spatiale et de texture : un patch Ψ^p appartenant à une région cohérente en termes de texture doit pouvoir être reconstruit à partir des patchs environnants : $\Psi^p = \sum_{\Psi^{p_k} \in \mathbf{\Psi} \setminus \Psi^p} w_k \Psi^{p_k}$.

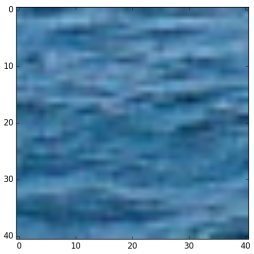
Dans la suite, nous allons considérer qu'une grande partie de l'image n'a pas de pixels manquant. Les patchs issus de cette partie de l'image constitueront notre dictionnaire d'atomes $\mathbf{\Psi}$ sur lesquels porteront nos combinaisons linéaires afin de reconstituer les parties manquantes.

1. Les 3 canaux sont usuellement le pourcentage de rouge, vert et bleu (rgb) ou sous format teinte, saturation, luminosité (hsv), plus proche de notre perception visuelle

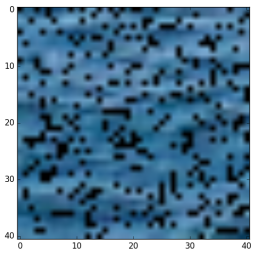
Supposons qu'un certain nombre des pixels d'un patch soient manquants : soit I l'ensemble des entrées manquantes correspondantes dans le vecteur Ψ^p et I^c son complémentaire. Nous noterons $\Psi^p_{I^c}$ l'ensemble des pixels exprimés du patch. La reconstruction consiste à faire l'hypothèse que si une combinaison linéaire est performante pour approximer $\Psi^p_{I^c}$, elle sera capable de généraliser aux valeurs manquantes Ψ^p_I. Le problème d'optimisation consiste dans ce cas à trouver le vecteur $\hat{\mathbf{w}}$ tel que :

$$\hat{\mathbf{w}} = \text{argmin}_{\mathbf{w}} \|\Psi^p_{I^c} - \sum_{\Psi^k \in \mathbf{\Psi}} w_k \Psi^k_{I^c}\|_2^2 + \lambda \|\mathbf{w}\|_1$$

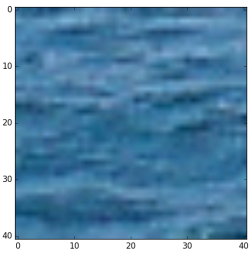
Autrement dit, les pixels présents sont utilisés pour l'apprentissage du vecteur $\mathbf{w}$ de régression et les valeurs manquantes sont les valeurs à prédire.



Patch origine



Patch bruité



Patch débruité

Télécharger le fichier `inpainting.py`. Il contient quelques fonctions utiles pour charger une image, extraire un patch, bruiter un patch, transformer un patch en vecteur et inversement, afficher un patch et construire un dictionnaire de patchs. Implémenter une fonction `denoise(patch,dic)` qui permet de débruiter le patch donné en entrée en fonction du dictionnaire `dic`. Faites varier le coefficient de régularisation. Que remarquez vous pour les valeurs faibles et grandes de λ ?
Supposons maintenant que c'est toute une partie de l'image qui est manquante. En considérant des patchs centrés sur les pixels manquants, on commence par remplir en partant des bords puis en remplissant au fur et à mesure vers le centre de l'image. A votre avis, l'ordre de remplissage a-t-il une importance ? Implémenter une fonction qui permet de compléter l'image. Proposer des heuristiques pour remplir de manière intelligente (voir [3]).

Références

[1] Bin Shen and Wei Hu and Zhang, Yimin and Zhang, Yu-Jin, *Image Inpainting via Sparse Representation* Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '09)

[2] Julien Mairal *Sparse coding and Dictionnary Learning for Image Analysis* INRIA Visual Recognition and Machine Learning Summer School, 2010

[3] A. Criminisi, P. Perez, K. Toyama *Region Filling and Object Removal by Exemplar-Based Image Inpainting* IEEE Transaction on Image Processing (Vol 13-9), 2004