TP4

Unknown Author

May 27, 2015

1 Apprentissage non supervisé

Ce domaine (unsupervised learning, clustering) regroupe un ensemble varié de techniques qui visent à trouver des sous-ensembles cohérents des données. Beaucoup d'approches existent selon les représentations considérées (similarité de données, représentation vectorielle, graphes), deux formulations principales sont généralement considérées étant un ensemble de données D :

- Clustering hard : on cherche une partition de D en k parties, deux à deux disjointes, qui minimise une certaine fonction de coût.
- Clustering soft : à chaque exemple on associe une probabilité d'appartenance à chaque cluster.

2 Apprentissage de prototype : Algorithme k-means

Un premier type d'approche pour le clustering est géométrique : il s'agit de trouver une partition de l'espace d'entrée en considérant la densité d'exemples pour caractériser ces partitions. Un exemple de tel algorithme est k-means : cet algorithme considère pour chaque cluster C_i un prototype $\mu_i \in \mathbb{R}^d$ dans l'espace d'entrée. Chaque exemple x est affecté au cluster le plus proche au sens de la distance euclidienne à son prototype. 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 $D = \{x_1, \ldots, x_n\}$ 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) dénotant le mélange de composantes rouge, vert et bleu. En limitant le nombre de couleurs possibles dans l'image (ce qu'on appelle un dictionnaire), on réalise une grosse compression. L'objectif est de trouver quelles couleurs doivent être présentent dans notre dictionnaire afin de minimiser l'erreur entre l'image compressé et l'image original (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 badse 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).

En codant vous même l'algorithme ou en utilisant la version de sklearn, expérimenter la compression : choisisser une image, construiser avec k-means l'image compressée et afficher là. Etudier en fonction du nombre de clusters (couleurs) choisis comment évolue l'erreur de reconstruction.

Quel est le gain en compresssion effectué ? est-il possible de gagner plus ?

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 ?

In [2]:
```
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
```

```
---------------------------------------------------------------------------
IOError                                   Traceback (most recent
call last)

<ipython-input-2-37379be41486> in <module>()
      1 import matplotlib.pyplot as plt
      2
----> 3 im=plt.imread("fichier.png")[:,:,:3] #on garde que les 3
premieres composantes, la transparence est inutile
      4 im_h,im_l,_=im.shape
      5 pixels=im.reshape((im_h*im_l,3)) #transformation en
matrice n*3, n nombre de pixels

/usr/lib/pymodules/python2.7/matplotlib/pyplot.pyc in
imread(*args, **kwargs)
   2175 @docstring.copy_dedent(_imread)
   2176 def imread(*args, **kwargs):
-> 2177     return _imread(*args, **kwargs)
   2178
   2179

/usr/lib/pymodules/python2.7/matplotlib/image.pyc in
imread(fname, format)
   1253     # tricky in C.
   1254     if cbook.is_string_like(fname):
-> 1255         with open(fname, 'rb') as fd:
   1256             return handler(fd)
   1257     else:
```

```
IOError: [Errno 2] No such file or directory: 'fichier.png'
```

3 Clustering spectral

Dans ce formalisme, on considère une fonction de similarité $s : X \times X \to \mathbb{R}^+$ entre les exemples : une représentation vectorielle n'est plus indispensable, des noyaux peuvent être utilisés. Cette fonction de similarité induit une structure de graphe complet ou non pondéré $G = (V, W)$ entre les exemples, ou les noeuds sont les exemples $V = \{x_i\}$ et les liens de pois la similarité entre deux exemples $W = \{w_{ij} = s(x_i, x_j)\}$. Le problème du clustering est équivalent à un problème classique de coupe dans un graphe. Une coupe C est définie par la donnée d'une partition en 2 ensembles des noeuds du graphe, son coût est $cut(C_1, C_2) = \sum_{i \in C_1, j \in C_2} w_{ij}$ et son coût normalisé : $cutN(C_1, C_2) = \frac{cut(C_1,C_2)}{volume(C_1)} + \frac{cut(C_1,C_2)}{volume(C_2)}$, avec $volume(C) = \sum_{i,j \in C} w_{ij}$. L'objectif est de trouver la coupe de coût minimal (pourquoi ?).

Ce problème est NP-hard, mais peut être relaxé et traiter algébriquement. Soit $M = \{w_{i,j}\}$ la matrice de similarité de ce graphe, symétrique, et L sa matrice laplacienne : $L = M - diag(d_i = \sum_j w_{ij})$.

Soit pour une coupe (C_1, C_2), soit f le vecteur dans $\{-1, 1\}^n$ tel que $f_i = 1$ si $i \in C_1$, $f_i = -1$ sinon. Alors $f' \not\Vdash f = \sum_{i,j} w_{ij}(f_i - f_j)^2$, ce qui est exactement le coût de la coupe. Ainsi, trouver f minimisant $\frac{f'Lf}{f'Df}$ tel que $f'D \not\Vdash = 0$ est équivalent au problème de coupe minimal. On relaxe le problème en considérant un vecteur réel et non pas entier et en fixant $f'Df = 1$. Le deuxième vecteur propre de L donne la solution. Cette étape donne un premier partitionnement. De manière récursive d'autres clusters peuvent être obtenus.

Sur la base de données movielens de la dernière fois, proposer des solutions pour le clustering des films, en fonction de leurs caractéristiques et/ou en fonction des votes et/ou des tags. Visualiser les matrices de similarités dans l'ordre aléatoire initiale. Comment la matrice de similarité permet d'indiquer visuellement un bon ou mauvais clustering ? Utiliser le clustering spectral et le clustering k-means et visualiser les matrices de similarités résultantes.

In []: