Intelligence artificielle & Apprentissage
TP apprentissage non supervisé : K-means et spectral clustering

F. Lauer

Pré-requis

Les algorithmes de clustering sont implémentés en python dans le module `clustering` de la bibliothèque scikit-learn (`sklearn`). Ils peuvent être importés par exemple avec

```
from sklearn.cluster import KMeans
from sklearn.cluster import SpectralClustering
```

Les méthodes de clustering s'utilisent ensuite ainsi :

```
clustering = NomMethode(n_clusters = K, ...)    # réglage des paramètres
clustering.fit(X)                               # applications aux données
```

où

$$\mathtt{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_m^T \end{bmatrix}$$

contient la base d'apprentissage.

Le résultat est ensuite récupéré avec

```
y = clustering.labels_
```

sous la forme d'un vecteur de taille m, où y_i est l'étiquette de l'exemple $\boldsymbol{x}_i$.

Pour les K-means, les centres des groupes sont obtenus avec

```
centres = clustering.cluster_centers_
```

et pour les méthodes spectrales, la matrice d'affinités (ou d'adjacence du graphe) avec

```
A = clustering.affinity_matrix_
```

1 Test de l'algorithme dans le plan

1.1 K-means

Remarque : Le comportement par défaut de `KMeans` n'est pas exactement celui vu en cours (quelques astuces sont utilisées pour améliorer les résultats). Pour retrouver ce comportement, il faut utiliser

```
clustering = KMeans(n_clusters=K, init='random', n_init=1 )
```

Dans le programme `clustering.py` :

1. Générez deux groupes de points bien séparés en 2D avec la souris et approximativement selon des distributions gaussiennes.

2. Complétez la fonction `monprogramme()` pour appliquer l'algorithme K-means à ces points avec K réglé par les touches +/− du clavier. Tracez ensuite les points en couleur sur le graphique pour observer la classification obtenue. Ajoutez également l'affichage des centres des groupes (avec un autre type de points).
3. Testez avec différentes valeurs de K.
4. Idem mais en créant plus de groupes de points plus ambigus (moins bien séparés). Observez l'aspect aléatoire du résultat.
5. Pour réduire l'influence de l'initialisation aléatoire sur le résultat, il est habituel de relancer l'algorithme à partir de plusieurs initialisations et de ne retenir que la meilleure. Pour cela, il suffit de spécifier le nombre d'initialisations avec le paramètre `n_init` dans

   ```
   clustering = KMeans(n_clusters=K, init='random', n_init = ... )
   ```

   Testez par exemple avec 100 initialisations. Comment la meilleure de toutes les initialisations est-elle déterminée ?

1.2 Spectral clustering

Remarque : Le comportement par défaut de `SpectralClustering` est celui vu en cours avec le laplacien normalisé $\boldsymbol{L}_{sym}$ et une matrice d'affinités $\boldsymbol{A}$ calculée avec un noyau gaussien : $A_{ij} = \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/2\sigma^2) = \exp(-\gamma\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2)$ de paramètre $\gamma = 1/2\sigma^2$ (`gamma`). Pour des affinités basées sur les n-plus proches voisins, il faut utiliser les paramètres `affinity='nearest_neighbors'` et `n_neighbors=n`.

Dans la fonction `monprogrammeSpectralClustering()` de `clustering.py` (exécutée en appuyant sur la touche `C`) :

1. Appliquez le spectral clustering aux données et affichez le résultat en couleurs sur le graphique. Affichez aussi dans une seconde figure la matrice d'affinités $\boldsymbol{A}$ comme une image en niveaux de gris avec `plt.imshow(A, cmap='gray')`.
2. Testez avec différentes valeurs de K et comparez les résultats à la méthode K-means. Essayez en particulier des distributions de points pour lesquelles la méthode K-means ne peut pas donner la bonne solution. Pensez aussi à tester différentes valeurs de σ (via `gamma` et `Ctrl +/-`) et observez son influence sur le résultat et la matrice $\boldsymbol{A}$.
3. Essayez d'implémenter la version de l'algorithme basé sur le laplacien normalisé $\boldsymbol{L}_{sym}$ dans une nouvelle fonction et comparez les résultats à ceux obtenus par `scikit-learn`.
   Ici, $\boldsymbol{D}^{-1/2}$ peut être calculée avec `np.sqrt(np.linalg.inv(D))` *avec* `D = np.sum(A, axis=1)`. *La matrice $\boldsymbol{A}$ peut être calculée avec la fonction* `kernel` *des TP précédents (sans oublier de mettre sa diagonale à 0, par exemple avec* `A = A - np.diag(np.diag(A))`*).*

2 Application à la segmentation d'images

Dans cette partie du TP, nous allons utiliser le clustering sur les pixels d'une image pour la segmenter, c'est-à-dire la découper en K régions de couleurs homogènes pouvant correspondre aux différents objets qui la composent.

Le programme `segmentation.py` permet de charger deux images et d'appliquer un traitement sur chacune d'elle dans une boucle `for` dans laquelle l'image est stockée dans `img` sous la forme d'un tableau à 3 dimensions. La 3ème dimension correspond aux canaux de couleurs rouge, vert, bleu. Ainsi, `img[:,:,0]` est une matrice représentant la composante rouge de l'image, `img[:,:,1]` la composante verte et `img[:,:,2]` la composante bleue ; et `img[i,j,:]` est le vecteur des 3 composantes de couleurs pour le pixel en position `i,j`.

1. Créez une matrice `X` au bon format pour pouvoir classer les pixels en groupes de couleurs similaires avec K-means. Vous pourrez utiliser `img.reshape( nbLignes, nbColonnes )`.
2. Appliquez K-means à cette matrice et récupérez les étiquettes sous forme d'une matrice `Y` de la même taille que l'image (en utilisant encore une fois `reshape`). Affichez cette matrice comme une image pour visualiser le découpage en régions obtenu. Les couleurs sont pour le moment arbitraires et choisies par `imshow`.

3. Testez pour différentes valeurs de K.
4. Nous souhaitons maintenant créer une image `segmentation` qui permettrait de voir les régions avec leur couleur moyenne correspondant au centre de chaque groupe. Complétez le programme pour remplir l'image `segmentation` (tableau à 3 dimensions) avec les bonnes couleurs et affichez-la. En plus de la segmentation, cela nous donne une discrétisation/quantification de l'image : un version basée sur K couleurs au lieu de 16 millions. Ceci permet par exemple de compresser les images, ou simplement de créer un effet "dessiné".