XAI - TME 9

Explication de données : clustering

L'objectif du TME est de comparer diverses méthodes de clustering, comme toujours sur des données en deux dimensions d'abord, puis sur des données de plus grande dimension.

Noter que les algorithmes de clustering par partitionnement font l'hypothèse que les données sont numériques.

`sklearn` propose des implémentations des algorithmes des k-moyennes, du clustering spectral, du clustering hiérarchique et de DBSCAN. Leurs fonctions et paramètres principaux sont rappelées en fin de sujet.

1. Implémenter les algorithmes des c-moyennes floues et possibilistes.
2. Comparer les résultats obtenus par les algorithmes de clustering sur des données artificielles constituées d'un nombre connu de gaussiennes, en faisant varier les paramètres des algorithmes

   En particulier, pour les GMM, les c-moyennes floues et possibilistes, tracer les lignes de niveaux des fonctions d'affectation pondérée.
3. Comparer les résultats obtenus sur les données `halfmoons` (clusters non gaussiens).
4. Pour étudier la robustesse des algorithmes par rapport aux exceptions, générer des données artificielles contenant deux clusters sphériques et une exception située à égale distance (cf transparent sur les c-moyennes possibilistes).

   Tracer l'ordonnée des centres de clusters identifiés en fonction de l'ordonnée de l'exception.

   Tracer également l'ordonnée des centres des clusters identifiés en fonction d'un nombre de données situées autour de l'exception (qui devient donc de moins en moins exceptionnelle).
5. Examiner la stabilité des algorithmes, c'est-à-dire leur sensibilité par rapport à une initialisation aléatoire.

Clustering dans `sklearn`

Pour chacun des algorithmes, il faut construire une instance de l'algorithme, nommée ici `algo` à titre d'exemple, puis l'appliquer aux données par la méthode `algo.fit(X)` où `X` est un `np.array`.

On rappelle ici les paramètres principaux des algorithmes disponibles.

k-moyennes

`sklearn.cluster.KMeans(n_clusters=k, init='random', n_init=10, max_iter=300, tol=0.0001, verbose=0, random_state=None)`

`init` peut prendre aussi la valeur `'k-means++'` : les centres sont alors initialisés selon une distribution de probabilité empirique des données. C'est une approche plus coûteuse, mais qui accélère la convergence.

`n_init` donne le nombre d'initialisations aléatoires considérées. Les résultats associés à la meilleure en terme de fonction de coût sont renvoyés.

`max_iter` et `tol` déterminent le critère d'arrêt, selon le nombre d'itération et la convergence en terme de stabilisation des positions des centres.

`random_state` peut être fixé à une valeur entière qui détermine les valeurs aléatoires générées.

Résultats : les affectations des points sont alors stockées dans `algo.labels_` et les centres dans `algo.cluster_centers_`

GMM

`sklearn.mixture.GaussianMixture(n_components=1, covariance_type='full', tol=0.001, max_iter=100, n_init=1, init_params='kmeans', random_state=None, verbose=0)`

`n_components` nombre de composantes (donc de clusters)

`covariance_type` : `full` cas général, `tied` une seule matrice de covariance partagée par toutes les composantes, `diag` matrices de covariance diagonales, `spherical`

`init_params` : `'kmeans', 'k-means++', 'random', 'random_from_data'`

Résultats : les centres sont stockées dans `algo.means_` et les matrices de covariance dans `algo.covariances_`

Clustering hiérarchique

`sklearn.cluster.AgglomerativeClustering(n_clusters=2, linkage='ward', distance_threshold=None)`

`linkage` stratégie de chaînage utilisée `'ward', 'complete', 'average', 'single'`.

`distance_threshold` : valeur de distance au dessus de laquelle les clusters ne sont pas fusionnés. Si une valeur est donnée, `n_clusters` doit valoir `None`.

Pour tracer un dendrogramme, voir le code fourni `https://scikit-learn.org/stable/auto_examples/cluster/plot_agglomerative_dendrogram.html`

Clustering spectral

`sklearn.cluster.SpectralClustering(n_clusters=8, random_state=None, n_init=10, gamma=1.0, affinity='rbf', degree=3, coef0=1)`

`random_state` et `n_init` comme pour les k-moyennes.

`gamma` paramètre du noyau pour les types `rbf, sigmoid, laplacian` ; `degree` et `coef0` paramètres pour le noyau polynomial.

`affinity` méthode de construction de la matrice d'affinité : `'rbf'` ou un noyau de type `pairwise_kernels`.

DBSCAN `sklearn.cluster.DBSCAN(eps=0.5, min_samples=5)`

`eps` distance maximale entre deux données pour qu'elles soient considérées dans le voisinage l'une de l'autre.

`min_samples` nombre de points du voisinage minimal pour être un *core point*

Résultat : les affectations des points sont alors stockées dans `algo.labels_`