TP Clustering
5e SDBD

MJ. HUGUET
homepages.laas.fr/huguet

Objectifs

Le but de ces TP est mettre en oeuvre et de comparer différents algorithmes de clustering tout d'abord à partir de quelques méthodes fournies par `scikit`***learn*** ou en utilisant des méthodes externes. Le planning des séances est le suivant :

- TP1 : méthodes k-Means et k-medoids
- TP2 : méthodes de clustering hiérarchique agglomératif et `DBSCAN`
- Travail personnel : `HDBSCAN` et préparation du rapport

Nous utilisons des jeux de données en 2 dimensions seulement pour des raisons pédagogiques. En effet, en visualisant ces exemples, il est souvent assez évident de déterminer le bon nombre de clusters à obtenir.

Encadrants

Marie-José Huguet, Mohamed Siala, Tim Luchterhand

Travail préparatoire

Lors des TP, vous allez développer en `Python` et utiliser `Anaconda` qui est la distribution Python installée à l'INSA. Avant de démarrer les séances, nous vous demandons de configurer votre environnement de travail. Cela devrait être rapide (maximum 15 minutes).

Sur une machine de TP, connectez-vous sur une session Linux. Depuis un terminal, vous pouvez lancer l'interface graphique d'`Anaconda` avec la commande `anaconda-navigator`.
L'environnement `Python` par défaut est celui appelé `base`. Pour éviter les conflits entre des versions de packages , vous allez créer un environnement virtuel pour chaque enseignement. Depuis l'interface graphique, allez sur l'onglet `Environnements`, puis créez un nouvel environnement appelé `clustering-tp`. Placez-vous dans cet environnement et ajoutez les packages `numpy, scipy, matplotlib, scikit-learn`. Vous pouvez remarquer que `VS Code` est présent dans votre environnement `clustering-tp`, nous vous conseillons de l'utiliser pour les codes Python à réaliser. Voilà, vous êtes prêts !

Si vous préférez créer votre environnement `Python` en ligne de commande, voila quelques instructions à exécuter depuis un terminal.

- `conda create -name clustering-tp` : crée un nouvel environnement appelé `clustering-tp`
- `conda activate clustering-tp` : vous place dans cet environnement. Tapez `conda activate` pour revenir à l'environnement `base`.
- `conda info -envs` : donne la liste des environnements disponibles
- `conda install numpy scipy matplotlib scikit-learn`
- `conda list` : fournit la liste des packages `Python` dans l'environnement courant

Le travail est à réaliser en binôme. Utilisez un dépot `git` de votre choix pour partager le code entre vous et pour le communiquer aux enseignants (un lien vers votre code Python sera demandé pour l'évaluation).

1 Jeux de données

Récupérez l'archive des jeux de données sur la page moodle des TP et placez les jeux de données dans un dossier appelé `artificial`. Les jeux de données proposés sont issus du site : `https://github.com/deric/clustering-benchmark` sur lequel vous pouvez les visualiser. Seuls les jeux de données "artificiels" seront considérés dans ces TP.

Récuperez également l'archive des codes `Python`, vous piocherez dedans en fonction de vos besoins.

Le code `1-Reading-arff-dataset.py` (présenté ci-dessous) fournit un exemple pour lire ces jeux de données et les visualiser en deux dimensions. Pour la lecture des jeux de données, il utilise le package `arff` de `from scipy.io`. Pour l'affichage, il utilise le package `pyplot` de `matplotlib`. N'hésitez pas à proposer des visualisations plus avancées.

```
import numpy as np
import matplotlib.pyplot as plt

from scipy.io import arff

# READ a data set (arff format)
# Parser un fichier de donnees au format arff
# datanp est un tableau (numpy) d'exemples avec pour chacun la liste
# des valeurs des features
#  Exemple : t =np.array([[1,2], [3,4], [5,6], [7,8]])
#
# Note  :
# les jeux de donnees consideres ont seulement 2 features (dimension 2 seulement)
# chaque jeu de donnees contient aussi un numero de cluster : on ignore

path = './artificial/'
name="square1.arff"
#path_out = './fig/'

databrut = arff.loadarff(open(path+str(name), 'r'))
datanp = np.array([[x[0],x[1]] for x in databrut[0]])

# PLOT donnees (en 2D) avec un scatter plot
# Extraire chaque valeur des features pour en faire une liste
# Exemple :
# - pour t1=t[:,0] --> [1, 3, 5, 7]
# - pour t2=t[:,1] --> [2, 4, 6, 8]
print("----------------------------------------")
print("Recuperer les donnees initiales            "+ str(name))
f0 = datanp[:,0] # tous les elements de la premiere colonne
f1 = datanp[:,1] # tous les elements de la deuxieme colonne
#print(f0)
#print(f1)

plt.figure(figsize=(6, 6)) # dimension de la figure
plt.scatter(f0, f1, s=10)  # s pour regler l'affichage des points
plt.title("Donnees initiales : "+ str(name))
#plt.savefig(path_out+"Plot-kmeans-code1-"+str(name)+"-init.jpg",bbox_inches='tight',
                                        pad_inches=0.1)
plt.show()
```

Dans les jeux de données `arff`, pour chaque exemple, la dernière colonne fournit le numéro de cluster (sans précision sur la méthode utilisée pour l'obtenir). En pratique, **vous ne devez pas utiliser cette colonne** car on suppose que les clusters ne sont pas connus.

2 Clustering k-Means

2.1 Pour démarrer

Le code fourni `2-Starting-with-k-means.py` permet d'appeler la méthode k-Means avec un nombre fixé de clusters et d'afficher le résultat ainsi que le temps de calcul, l'inertie totale et le nombre d'itérations.

```
import numpy as np
import matplotlib.pyplot as plt
import time
from sklearn import cluster

# Lecture fichier au format arff
# ici squarre1.arff comportant 4 clusters
# ......
# ......

# Les donnees sont dans datanp (2 dimensions)
# f0 : valeurs sur la premiere dimension
# f1 : valeur sur la deuxieme dimension
#
# Run clustering method for a given number of clusters
print("------------------------------------------------------")
print("Appel KMeans pour une valeur de k fixee")
tps1 = time.time()
k=4
model = cluster.KMeans(n_clusters=k, init='k-means++', n_init=1)
model.fit(datanp)
tps2 = time.time()
labels = model.labels_
# informations sur le clustering obtenu
iteration = model.n_iter_
inertie = model.inertia_
centroids = model.cluster_centers_

#plt.figure(figsize=(6, 6))
plt.scatter(f0, f1, c=labels, s=8)
plt.scatter(centroids[:, 0],centroids[:, 1], marker="x", s=50, linewidths=3,
                                            color="red")
plt.title("Donnees apres clustering : "+ str(name) + " - Nb clusters ="+ str(k))
#plt.savefig(path_out+"Plot-kmeans-code1-"+str(name)+"-cluster.jpg",bbox_inches='
                                            tight', pad_inches=0.1)
plt.show()

print("nb clusters =",k,", nb iter =",iteration, ", inertie = ",inertie, ",
                                            runtime = ", round((tps2 - tps1)*1000,
                                            2),"ms")
#print("labels", labels)
```

A l'aide des distances proposées dans **scikit*learn***, calculez :

- les scores de **regroupement** de chaque cluster (distance minimale, maximale, moyenne entre les points d'un cluster et son centre)
- les scores de **séparation** entre les clusters (distance minimale, maximale, moyenne entre les différents centres)

Visualisez les résultats obtenus sur les jeux de données `square1.arff` et `xclara.arff`.

2.2 Application itérative de la méthode k-Means basée sur l'inertie

Appliquez la méthode k-Means pour différentes valeurs de k. Pour chaque valeur, mémorisez la valeur de l'inertie de chaque clustering obtenu. Affichez l'évolution de l'inertie. Comment déterminer une bonne solution de clustering ? Quels sont les scores de regroupement et de séparation de la solution obtenue ?

2.3 Application itérative de la méthode k-Means basée sur des métriques d'évaluation externes

Différents indicateurs peuvent être employés pour évaluer le résultat d'un clustering. `scikit`***learn*** propose des implémentatations des métriques suivantes : coefficient de silhouette, indice de Davies-Bouldin et indice de Calinski-Harabasz) [1].

- Appliquez itérativement la méthode k-Means pour déterminer le bon nombre de clusters à l'aide d'une de ces métriques.
- Mesurez le temps de calcul (total)
- Quels sont les scores de regroupement et de séparation de la solution obtenue ?

2.4 Intérêts et Limites de la méthode k-Means

Afin d'illustrer les caractéristiques de la méthode k-Means dans votre rapport de TP, vous allez sélectionner quelques jeux de données.

- 2 ou 3 jeux de données pour lesquels vous pensez que la méthode k-Means devrait identifier correctement les clusters.
- 2 ou 3 jeux de données pour lesquels vous pensez que la méthode k-Means devrait avoir des difficultés.
- Appliquez la méthode k-Means (application incrémentale en vous basant sur le travail effectué dans les sections précédentes) pour confirmer vos choix.

Que retenez-vous de ces expérimentations ?

2.5 Version mini-batch de la méthode k-Means

Reprenez les expérimentations effectuées dans la section précédente et remplacez la méthode k-Means par la variante `minibatch` fournie dans `scikit`***learn***.

N'hésitez pas à tester différents paramétrages de ces méthodes.

Analysez vos résultats (pour le rapport de TP).

1. `https://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation`

3 Clustering agglomératif

3.1 Pour démarrer

Le code `3-Dendrogramme.py` fourni ci-dessous permet de calculer un clustering agglomératif et d'afficher le dendrogramme associé. La méthode de 'linkage' utilisée est `average`. Avez-vous retenu à quoi sert cette méthode ?

```
from scipy.cluster.hierarchy import dendrogram

def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack([model.children_, model.distances_, counts])
                                                    .astype(float)

    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix) #, **kwargs)


# setting distance_threshold=0 ensures we compute the full tree.
model = cluster.AgglomerativeClustering(distance_threshold=0, linkage='average',
                                        n_clusters=None)

model = model.fit(datanp)
plt.figure(figsize=(12, 12))
plt.title("Hierarchical Clustering Dendrogram")
# plot the top p levels of the dendrogram
plot_dendrogram(model) #, truncate_mode="level", p=5)
plt.xlabel("Number of points in node (or index of point if no parenthesis).")
plt.show()
```

Pour les jeux de données `square1.arff` et `xclara.arff`, testez différentes variantes de 'linkage'. Testez différents jeux de données et analysez les résultats obtenus. Vous pouvez considérer d'autres jeux de données.

3.2 Application de la méthode de clustering agglomérative

Le code `4-Starting-with-Agglomerative-Clustering.py` permet de déterminer un clustering hiérarchique en utilisant soit une limite sur le seuil de distance soit un nombre de clusters.

```
### FIXER la distance
tps1 = time.time()
model = cluster.AgglomerativeClustering(distance_threshold=10, linkage='average',
                                        n_clusters=None)
model = model.fit(datanp)
tps2 = time.time()
k = model.n_clusters_
leaves=model.n_leaves_
plt.scatter(f0, f1, c=labels, s=8)
plt.title("Clustering agglomeratif (average, distance_treshold=10) "+str(name))
plt.show()
print("nb clusters =",k,", nb feuilles = ", leaves, " runtime = ", round((tps2 -
                                        tps1)*1000,2),"ms")

# FIXER le nombre de clusters
k=4
tps1 = time.time()
model = cluster.AgglomerativeClustering(linkage='average', n_clusters=k)
model = model.fit(datanp)
tps2 = time.time()
kres = model.n_clusters_
leaves=model.n_leaves_

plt.scatter(f0, f1, c=labels, s=8)
plt.title("Clustering agglomeratif (average, n_cluster=4) "+str(name))
plt.show()
print("nb clusters =",kres,", nb feuilles = ", leaves, " runtime = ", round((tps2
                                        - tps1)*1000,2),"ms")
```

- Pour une méthode de linkage donnée, appliquez itérativement la méthode de clustering agglomératif pour déterminer le seuil de distance (ou le nombre de clusters) obtenant la meilleure valeur pour une des métriques d'évaluation étudiées précédemment.
- Mesurez le temps de calcul du clustering agglomératif itératif pour chaque méthode de linkage et pour l'ensemble des variantes de linkage testées.
- Quels sont les scores de regroupement et de séparation de la solution obtenue ?

3.3 Intérêts et Limites du clustering agglomératif

Afin d'illustrer les caractéristiques de la méthode clustering agglomératif dans votre rapport de TP, vous allez sélectionner quelques jeux de données.

- 2 ou 3 jeux de données pour lesquels vous pensez que le clustering agglémeratif devrait identifier correctement les clusters.
- 2 ou 3 jeux de données pour lesquels vous pensez que le clustering agglémeratif devrait avoir des difficultés.
- Appliquez clustering agglémeratif (application incrémentale de différents linkage en vous basant sur le travail effectué dans les sections précédentes) pour confirmer vos choix.

Que retenez-vous de ces expérimentations ?

4 Clustering DBSCAN et HDBSCAN

4.1 Pour démarrer

La première partie du code `5-Starting-with-DBSCAN.py` permet d'appeler la méthode `DBSCAN`sur le jeu de données `xclara1.arrf` et de fixer les paramètres `eps` et `min_samples` de cette méthode.

```
# Run DBSCAN clustering method
# for a given number of parameters eps and min_samples
#
print("------------------------------------------------------")
print("Appel DBSCAN (1) ... ")
tps1 = time.time()
epsilon=2
min_pts=5
model = cluster.DBSCAN(eps=epsilon, min_samples=min_pts)
model.fit(datanp)
tps2 = time.time()
labels = model.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
n_noise = list(labels).count(-1)
print('Number of clusters: %d' % n_clusters)
print('Number of noise points: %d' % n_noise)

plt.scatter(f0, f1, c=labels, s=8)
plt.title("Donnees apres clustering DBSCAN (1) - Epislon= "+str(epsilon)+" MinPts
                                        = "+str(min_pts))
plt.show()
```

Testez différentes valeurs de ces paramètres afin de déterminer un nombre de clusters qui vous semble pertinent.

Il est également possible de standardiser les données avant de rechercher des valeurs de ces paramètres. La seconde partie du code `5-Starting-with-DBSCAN.py` permet d'effectuer cette standardisation.

4.2 Application de la méthode DBSCAN

Afin de rechercher des paramètres pertinents pour les paramètres `eps` et `min_samples` on peut calculer les distances moyenne aux k plus proches voisins pour chaque exemple d'un jeu de données et déterminer le "coude" de la courbe.

```
# Distances aux k plus proches voisins
# Donnees dans X
k=5
neigh = NearestNeighbors(n_neighbors=k)
neigh.fit(X)
distances, indices = neigh.kneighbors(X)

# distance moyenne sur les k plus proches voisins
# en retirant le point "origine"
newDistances = np.asarray([np.average(distances[i][1:]) for i in range(0,
                                        distances.shape[0])])
# trier par ordre croissant
distancetrie = np.sort(newDistances)

plt.title("Plus proches voisins "+str(k))
plt.plot(distancetrie);
plt.show()
```

En utilisant cette méthode, appliquer la méthode `DBSCAN` sur quelques jeux de données et évaluez la qualité des solutions obtenues.

4.3 Intérêts et Limites de la méthode `DBSCAN`

Afin d'illustrer les caractéristiques de la méthode `DBSCAN` dans votre rapport de TP, vous allez sélectionner quelques jeux de données.

- 2 ou 3 jeux de données pour lesquels vous pensez que `DBSCAN` devrait identifier correctement les clusters.
- 2 ou 3 jeux de données pour lesquels vous pensez que `DBSCAN` devrait avoir des difficultés.
- Appliquez `DBSCAN` en vous basant sur le travail effectué dans les sections précédentes pour confirmer vos choix.

Que retenez-vous de ces expérimentations ?

4.4 Comparaison avec la méthode `HDBSCAN`

Le package `Python` de cette méthode ainsi que sa documentation sont accessibles ici [2] . Récupérez le avec la commande `conda install -c conda-forge hdbscan`. Vous pouvez également avoir besoin auparavant d'installer le package cython (`conda install cython`).

La méthode `HDBSCAN` dispose de deux paramètres principaux `min_samples` et `min_cluster_size` que vous allez tester sur différents jeux de données.

Cette méthode est connue pour être insensible à la variabilité de densité dans les données. Pour votre rapport, reprenez les expérimentations effectuées avec `DBSCAN`. Comparez les résultats de ces deux méthodes. Arrivez-vous à retrouver les qualités et les limites de ces deux méthodes sur les jeux de données sélectionnés ? Y-a-t-il des différences de performances ?

5 Evaluation

Ces TP sont évalués par un rapport du travail réalisé. Le plan du rapport est le suivant :

- Section 1 : introduction contenant le lien vers votre code (dépôt git)
- Section 2 à 4 : suivre le plan du sujet de TP pour répondre aux différentes attentes
- Section 5 : déterminer des solutions de clustering pour des jeux de données complémentaires fournis sur moodle
- Section 6 : conclusion

Le rapport est à déposer sur moodle au format pdf (un rapport par binôme). La date limite est précisée sur la page moodle.

Pour lancer votre analyse expérimentale vous pouvez utiliser différents serveurs de calcul (normalement accessible à distance via le vpn insa) : srv-ens-calcul ou srv-gei-gpu1 et srv-gei-gpu2.

2. `https://hdbscan.readthedocs.io` et `https://github.com/scikit-learn-contrib/hdbscan`