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, Julien Ferry, Rafael Bailon-Ruiz

Environnement de travail

Dans les salles de TP, connectez vous sur une session Linux. Dans un terminal, vous pouvez lancer `Anaconda` avec la commande `anaconda-navigator`.
Depuis `Anaconda`, créez un environnement spécifique pour les TP de clustering. Dans cet environnement, installez `Spyder` qui sera votre IDE Python.

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).

Pour installer les packages Python nécessaires, ouvrez un terminal. Entrez la commande `conda init` pour mettre à jour les variables d'environnement système. Fermez le terminal.
Pour pouvoir ensuite basculer sur votre environnement virtuel, ouvrez un terminal puis tapez la commande `conda activate nom_environnement`. Vous pouvez ensuite installer les packages Python nécessaires pour ces TP avec la commande `pip install`. Dans un premier temps vous aurez besoin des packages `scipy, numpy, matplotlib, scikit-learn`.

1 Jeux de données

Les jeux de données sont disponibles sur le site : `https://github.com/deric/clustering-benchmark`. Seuls les jeux de données "artificiels" seront considérés dans ces TP (`https://github.com/deric/clustering-benchmark/tree/master/src/main/resources/datasets/artificial`).

Le code 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

# Parser un fichier de donnees au format arff
# data est un tableau d'exemples avec pour chacun
#       la liste des valeurs des features
#
# Dans les jeux de donnees consideres :
#   il y a 2 features (dimension 2)
# Ex : [[-0.499261, -0.0612356],
#       [-1.51369, 0.265446],
#       [-1.60321, 0.362039], .....
#      ]
#
# Note : chaque exemple du jeu de donnees contient aussi un
# numero de cluster. On retire cette information

path = './artificial/'
databrut = arff.loadarff(open(path+"xclara.arff", 'r'))
data = [[x[0],x[1]] for x in databrut[0]]

# Affichage en 2D
# Extraire chaque valeur de features pour en faire une liste
# Ex pour f0 = [-0.499261, -1.51369, -1.60321, ...]
# Ex pour f1 = [-0.0612356, 0.265446, 0.362039, ...]
f0 = [f[0] for f in data]
f1 = [f[1] for f in data]

plt.scatter(f0, f1, s=8)
plt.title("Donnees initiales")
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 et k-Medoids

2.1 Pour démarrer

Le code ci-dessous permet d'appeler la méthode k-Means avec un nombre fixé de clusters et d'afficher le résultat ainsi que le temps de calcul et le nombre d'itérations.

```
import numpy as np
import matplotlib.pyplot as plt
import time
from sklearn import cluster
#
# Les donnees sont dans datanp (2 dimensions)
# f0 : valeurs sur la premiere dimension
# f1 : valeur sur la deuxieme dimension
#
print("Appel KMeans pour une valeur fixee de k ")
tps1 = time.time()
k=3
model = cluster.KMeans(n_clusters=k, init='k-means++')
model.fit(datanp)
tps2 = time.time()
labels = model.labels_
iteration = model.n_iter_

plt.scatter(f0, f1, c=labels, s=8)
plt.title("Donnees apres clustering Kmeans")
plt.show()
print("nb clusters =",k,", nb iter =",iteration, ", ...
... runtime = ", round((tps2 - tps1)*1000,2),"ms")
```

2.2 Intérêts de la méthode k-Means

Choisissez quelques (2 ou 3) jeux de données pour lesquels il vous semble que la méthode k-Means devrait identifier correctement les clusters.

On considère qu'il peut être possible de déterminer "automatiquement" le bon nombre de clusters. Utilisez les métriques d'évaluation proposées dans scikitlearn (coefficient de silhouette et/ou indice de Davies-Bouldin et/ou l'indice de Calinski-Harabasz)[1].

- Appliquez itérativement la méthode précédente pour déterminer le bon nombre de clusters à l'aide de métriques d'évaluation
- Mesurez le temps de calcul
- Arrivez-vous à retrouver le résultat attendu à l'aide de ces métriques d'évaluation ?

2.3 Limites de la méthode k-Means

Choisissez quelques (2 ou 3) jeux de données pour lesquels il vous semble que la méthode k-Means aura des difficultés pour identifier correctement les clusters.

Appliquez la méthode k-Means (en faisant varier la valeur de k) sur ces jeux de données pour confirmer vos choix.

2.4 Méthode k-medoids

Plusieurs méthode de clustering k-medoids sont disponibles en récupérant le package `kmedoids`. Le site `https://python-kmedoids.readthedocs.io/` fournit une documentation de ces méthodes. Un exemple d'utilisation (pour une valeur fixée de k et pour une métrique de distance donnée) est fourni ci-dessous.

```
from sklearn import metrics
import kmedoids
```

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

```
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.metrics.pairwise import manhattan_distances

tps1 = time.time()
k=3
distmatrix = euclidean_distances(datanp)
fp = kmedoids.fasterpam(distmatrix, k)
tps2 = time.time()
iter_kmed = fp.n_iter
labels_kmed = fp.labels
print("Loss with FasterPAM:", fp.loss)

plt.scatter(f0, f1, c=labels_kmed, s=8)
plt.title("Donnees apres clustering KMedoids")
plt.show()
print("nb clusters =",k,", nb iter =",iter_kmed, ", ...
... runtime = ", round((tps2 - tps1)*1000,2),"ms")
```

- Appliquez itérativement la méthode précédente pour déterminer le bon nombre de clusters à l'aide de métriques d'évaluation (en conservant la distance euclidenne)
- Mesurez le temps de calcul
- Arrivez-vous à retrouver le résultat attendu à l'aide de métriques d'évaluation fournies dans le package `kmedoids` ?
- Pour la valeur de k identifiée comme la plus pertinente, comparez les résultats obtenus par le clustering k-means et par le clustering k-medoids à l'aide d'indicateurs comme `rand_score` ou `mutual_information`.

Avec la méthode des k-medoids, en plus du nombre de clusters, on peut également faire varier la métrique de distance.

- Testez l'impact de la métrique de distance sur quelques exemples.

3 Clustering agglomératif

3.1 Pour démarrer

Le code ci-dessous permet d'afficher un dendrogramme (il y a d'autres possibilités ...) avec la méthode d'agglomération de clusters `single`.

```
import scipy.cluster.hierarchy as shc

# Donnees dans datanp
print("Dendrogramme 'single' donnees initiales")

linked_mat = shc.linkage(datanp, 'single')

plt.figure(figsize=(12, 12))
shc.dendrogram(linked_mat,
orientation='top',
distance_sort='descending',
show_leaf_counts=False)
plt.show()
```

Le code suivant permet de déterminer un clustering hiérarchique en utilisant soit une limite sur le seuil de distance soit un nombre de clusters.

```
# set distance_threshold (0 ensures we compute the full tree)
tps1 = time.time()
model = cluster.AgglomerativeClustering(distance_threshold=10, ...
... linkage='single', n_clusters=None)
model = model.fit(datanp)
tps2 = time.time()

labels = model.labels_
k = model.n_clusters_
leaves=model.n_leaves_
# Affichage clustering
plt.scatter(f0, f1, c=labels, s=8)
plt.title("Resultat du clustering")
plt.show()
print("nb clusters =",k,", nb feuilles = ", leaves, ...
... " runtime = ", round((tps2 - tps1)*1000,2),"ms")


# set the number of clusters
k=4
tps1 = time.time()
model = cluster.AgglomerativeClustering(linkage='single', n_clusters=k)
model = model.fit(datanp)
tps2 = time.time()

labels = model.labels_
kres = model.n_clusters_
leaves=model.n_leaves_

...
```

3.2 Intérêts de la méthode

Choisissez quelques (2 ou 3) jeux de données pour lesquels il vous semble que la méthode de clustering aggomératif devrait identifier correctement les clusters.

- Appliquez itérativement la méthode de clustering agglomératif en faisant varier le seuil de distance afin de déterminer une bonne solution de clustering à l'aide des métriques d'évaluation
  - Considérez différentes manières de combiner des clusters (single, average, complete, ward linkage), uniquement pour la distance euclidienne. Par défaut l'option `connectivity` est laissée à `none`.
  - Mesurez le temps de calcul
  - Arrivez-vous à retrouver le résultat attendu à l'aide de ces critères d'évaluation ?
- Vous avez automatiser votre code ? Recommencez en faisant varier le nombre de clusters

3.3 Limites de la méthode

Choisissez quelques (2 ou 3) jeux de données pour lesquels il vous semble que la méthode de clustering agglométarif aura des difficultés pour identifier correctement les clusters.

Appliquez la méthode de clustering agglomératif sur ces jeux de données pour confirmer vos choix.

3.4 Comparaison de méthodes de clustering

Proposez une comparaison des résultats obtenus par les méthodes k-means, k-medoids et clustering agglomératif.