Université Centrale	Module:Atelier Machine Learning & Data mining
Institut Télécom	Sections : Master-BD
Département Informatique	Niveau : 1ère année
Année Universitaire : 2019 - 2020	Enseignante : Sana Hamdi

TP N°4 :

Apprentissage non supervisé : Clustering

OBJECTIFS DU TP

- Programmer deux méthodes de clustering en nous appuyant sur deux procédures des packages spécialisés pour **Python**: la classification ascendante hiérarchique (CAH : Package SciPy) ; la méthode des centres mobiles (k-Means : Package Scikit-Learn).

L'objectif principal des méthodes d'apprentissage non supervisé est de grouper des éléments proches dans un même groupe d'une manière que deux éléments d'un même groupe soient le plus similaire possible et que deux éléments de deux groupes différents soient le plus dissemblables possible. On cherche donc à minimiser la distance intra-classe et de maximiser la distance inter-classe. Plusieurs méthodes sont disponibles dans Scikit- learn. Nous étudierons une d'entre elles de plus près dans ce TP.

Partie 1 : Données de travail

On dispose d'un jeu de données « fromage.txt » qui comprend 29 instances de fromage, représentées par 9 attributs (calories, sodium, calcium, lipides, retinol, folates, proteines, cholesterol et magnesium).

Les instructions Python suivantes permettent de charger le jeu de données « fromage.txt »

```
import pandas as pd
fromage=pd.read_table(r"…\fromage.txt",sep="\t",header=0,index_col=0)
print(fromage.describe())
```

A Vous !

1. Comprendre et programmer les quelques lignes Précédentes : comment sont réparties les données? Quels sont les attributs de ce jeu de données?
2. Maintenant on veut visualiser la matrice de distribution de données :

```
#croisement 2 à 2 des variables
pd.scatter_matrix(fromage, figsize=(9,9))
```

Qu'est-ce que vous remarquez ?

Partie 2 : Méthode des centres mobiles

Dans l'algorithme k-means, le nombre k de clusters est fixé au départ. A partir d'une partition initiale, on cherche à améliorer itérativement la partition en minimisant un certain critère.

Nous importerons le paquet **sklearn.cluster** pour utiliser la méthode k-means de la classe **sklearn.cluster.KMeans.** La description de l'implémentation de la méthode des K-moyennes (K-means) se trouve dans :

http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

L'exécution de l'algorithme k-means se fait par la commande suivante :

```
KMeans(n_clusters=8,init='k-means++',n_init=10,max_iter=300,
tol=0.0001, precompute_distances='auto', n_jobs=1)
```

- *n_clusters*: le nombre de classes (par défaut n_clusters =8).
- *init:{'k-means++','random'ou 'unndarray'}*: est une méthode d'initialisation, par défaut 'k-means++':
  - *'K-means ++' :* sélectionne intelligemment les centres initiaux afin d'accélérer la convergence.
  - *'Random'*: choisit k observations (rangées) au hasard parmi les données pour les centres initiaux.
  - *'ndarray'* : passe en paramètre les centres initiaux sous la forme (n_clusters, n_features).
- *n_jobs=1* permet d'exécuter les n_init itérations en parallèle.

cluster_centers_: contient les attributs en sortie : les centres

labels_: les numéros de cluster de chaque observation

inertie : la somme des distances au carré des observations vers leur centre de cluster le plus proche.

Les commandes suivantes permettent :

- `km = KMeans(n_clusters=n_cluster)` permet de créer un modèle pour un ensemble de k centres.
- L'instruction `km.fit(X)` utilise les données pour définir le modèle de clustering.
- `predict(X)` prédit le cluster le plus proche auquel appartient chaque échantillon.

A Vous !

1. Comprendre et programmer les lignes suivantes :

```
import numpy as np
from sklearn import cluster
np.random.seed(0)
kmeans = cluster.KMeans(n_clusters = 4)
kmeans.fit(fromage)
idk = np.argsort(kmeans.labels_)
print(pd.DataFrame(fromage.index[idk],kmeans.labels_[idk]))
print(kmeans.transform(fromage))
```

2. Donnez les attributs des centres de chaque cluster.

Aide à la détection du nombre adéquat de groupes

K-Means, ne fournit pas d'outils d'aide à la détection du nombre de classes. Nous devons les programmer sous Python ou utiliser des procédures proposées par des packages dédiés. Le schéma est souvent le même : on fait varier le nombre de groupes et on surveille l'évolution d'un indicateur de qualité de la solution c.-à-d. l'aptitude des individus à être plus proches de ses congénères du même groupe que des individus des autres groupes.
Dans ce qui suit, on calcule la métrique «silhouette» pour différents nombres de groupes issus de la méthode des centres mobiles.

A Vous !

1. Comprendre et programmer les lignes suivantes :

```
from sklearn import metrics
#utilisation de la métrique "silhouette"
res = np.arange(9, dtype ="double")
for k in np.arange(9):
    km = cluster.KMeans(n_clusters = k+2)
    km.fit(fromage)
    res[k] = metrics.silhouette_score(fromage,km.labels_)
print (res)
#graphique
import matplotlib.pyplot as plt
```

```
plt.title("silhouette")
plt.xlabel("# of clusters")
plt.plot(np.arange(2,11,1),(res))
plt.show
```

2. Quelle est la meilleure valeur de k (nombre de clusters) selon la métrique silhouette ?

Partie 3 : Classification ascendante hiérarchique

La classification ascendante hiérarchique (ou CAH) procède par fusions successives d'ensembles de points (clusters), en considérant initialement tous les points comme des clusters singletons, on fusionne à chaque étape les 2 clusters les plus proches au sens d'une distance, jusqu'à obtenir un seul cluster contenant tous les points.

Nous allons exécuter l'algorithme **CAH** en utilisant le package **scipy.**

A Vous !

1. Comprendre et programmer les lignes suivantes :

```
import pandas
import numpy as np
#librairies pour la CAH
from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
Z = linkage(fromage,method='ward', metric='euclidean')
#affichage du dendrogramme
plt.title("CAH")
plt.title('CAH avec matérialisation des 4 classes')
dendrogram(Z,labels=fromage.index,orientation='left',color_th
reshold=255)
plt.show()
groupes_cah = fcluster(Z, t = 255,criterion='distance')
print(groupes_cah)
#index triés des groupes
idg = np.argsort(groupes_cah)
#affichage des observations et leurs groupes
print(pandas.DataFrame(fromage.index[idg],groupes_cah[idg]))
```

2. Interpréter les résultats
3. Donnez la correspondance entre les groupes de la CAH et les clusters de Kmeans en utilisant la méthode `pandas.crosstab`

Partie 3 : Interprétation des classes

Analyse en composantes principales (ACP)

Dans Scikit-learn, l'analyse en composantes principales (ACP) est mise en oeuvre dans la classe `sklearn.decomposition.PCA` avec :

```
PCA(n_components=None,copy=True,whiten=False,svd_solver='auto,
tol=0.0, iterated_power='auto', random_state=None).
```

Les paramètres d'appel sont (pour des explications plus détaillées voir le lien ci-dessus) :

- **n_components** : nombre de composantes à déterminer (par défaut toutes, c'est à dire min(n_samples, n_features) où n_samples est le nombre de lignes de la matrice de données et n_features le nombre de colonnes).
- **copy** : si False, la matrice de données est écrasée par les données transformées (par défaut True).
- **whiten** : si True, les données transformées sont standardisées pour que les projections sur les axes principaux présentent une variance unitaire (par défaut False).

A Vous !

1. Comprendre et programmer les lignes suivantes :

```
from sklearn.decomposition import PCA
acp = PCA(n_components =2).fit_transform(fromage)
for couleur,k in zip(['red','blue','lawngreen', 'aqua'],[0,1,2,3]):
    plt.scatter(acp[km.labels_==k,0],acp[km.labels_==k,1],c=couleur)
plt.show()
```

2. Interprétez les résultats

Compte rendu :

1. Donnez le code (doit être richement commenté)
2. Appliquez sur le même jeu de données, la méthode de clustering CAH du package sklearn.cluster.AgglomerativeClustering. Comparez les résultats.
3. Implémentez un algorithme hiérarchique descendant (divisive clustering) basé sur l'utilisation de k-means