Les algorithmes de clustering

Exercice 1 :

1) Appliquer les centres mobiles à la main pour trouver une partition en deux classes des quatre points-individus ind1, ind2, ind3, ind4 en prenant ind1 et ind2 comme centres initiaux.

Id	Var1	Var2
ind1	5	4
ind2	4	5
ind3	1	-2
ind4	0	-3

Calculer l'inertie intra-classe de cette partition.

2) Recommencez avec le logiciel Python et vérifiez que vous retrouvez la même partition (en partant des mêmes deux centres initiaux) et que vous retrouvez l'inertie intra-classe calculée ci-dessus.

3) Calculer le pourcentage de l'inertie du nuage des 4 points-individus expliqué par cette partition en 2 classes.

Exercice 2 :

1) Appliquer à la main l'algorithme de classification hiérarchique sur le tableau de données de l'exercice 1 en prenant le lien maximum comme mesure d'agrégation. Quelle est alors la hiérarchie H et la hiérarchie indicée (représentée par le dendrogramme). Quelle est la partition en deux classes issue du dendrogramme du lien max ?

2) A la main toujours, faire le dendrogramme obtenu avec la mesure d'agrégation de Ward.

3) Avec les fonctions de Python, construire la hiérarchie indicées du lien maximum, son dendrogramme et la partition en deux classes associée. Vérifier que vousretrouvez les mêmes résultats qu'à la main.

4) Idem pour Ward.

Exercice 3 :

Programmer quelques méthodes de clustering, évaluer leurs performances, en utilisant la bibliothèque **scikit-learn** de Python.

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-classeet de maximiser la distance inter-classe. Plusieurs méthodes sont disponibles dans Scikit- learn. Nous étudierons quelques-unes d'entre elles de plus près dans ce TP.

Données de travail

On dispose d'un jeu de données digits 6 comprend 1797 images de résolution 8x8, représentant un chiffre manuscrit. Le champ image est un tableau 8 × 8 d'entiers représentant des niveaux de gris et variant de 0 (blanc) à 16 (noir). Le champ data est un vecteur de dimension 64 comprenant la même information. Le champ target est un chiffre compris entre 0 et 9.

Les instructions Python suivantes permettent de charger le jeu de données digits :

```
from sklearn.datasets import load_digits
digits=load_digits()
```

PARTIE 1 :

La méthode k-means

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.

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 un ndarray} :** 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 :

- La ligne **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.

L'Analyse en Composantes Principales

Dans Scikit-learn, l'analyse en composantes principales (ACP) est mise en œuvre 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).

http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html).

Travail à faire :

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. Ecrivez un programme TP1_prog1.py qui permet de :

1. Charger et manipuler le jeu de données digits,
2. Effectuer un clustering du jeu de données en utilisant l'algorithme K-Means et en conservant les paramètres par défaut.
3. Modifier l'attribut **init = 'random'.**
4. Appliquer une ACP sur le jeu de données digits avec un nombre de composantes = 8.
5. Afficher le groupe des images 0, 1, 2, 3 et les images correspondantes. Qu'est-ce que vous remarquez ?
6. Calculez le score des modèles construits à l'aide de la méthode de Silhouette Coefficient de la classe sklearn.metrics.silhouette_score. (http://scikit-learn.org/stable/modules/clustering.html
7. Faites varier le nombre de clusters de 1 à 15 et afficher la courbe des scores. Quel est le meilleur nombre de clusters K pour le jeu de données?
   - Utilisez une boucle.
   - Utilisez la méthode de Silhouette (Elbow Curve : interprétez cette courbe).
8. Exécutez et expliquez le code qui suit.
9. A votre avis, quels sont les avantages et les inconvénients de k-means

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

from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale

np.random.seed(42)

digits = load_digits()
data = scale(digits.data)
n_samples, n_features = data.shape
n_digits = len(np.unique(digits.target))
labels = digits.target

sample_size = 300

print("n_digits: %d, \t n_samples %d, \t n_features %d"
      % (n_digits, n_samples, n_features))

print(82 * '_')
print('init\t\ttime\tinertia\thomo\tcompl\tv-
meas\tARI\tAMI\tsilhouette')

def bench_k_means(estimator, name, data):
    t0 = time()
    estimator.fit(data)
    print('%-9s\t%.2fs\t%i\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f'
          % (name, (time() - t0), estimator.inertia_,
             metrics.homogeneity_score(labels, estimator.labels_),
             metrics.completeness_score(labels, estimator.labels_),
             metrics.v_measure_score(labels, estimator.labels_),
             metrics.adjusted_rand_score(labels, estimator.labels_),
             metrics.adjusted_mutual_info_score(labels,
estimator.labels_),
             metrics.silhouette_score(data, estimator.labels_,
                                      metric='euclidean',
                                      sample_size=sample_size)))

bench_k_means(KMeans(init='k-means++',          n_clusters=n_digits,
n_init=10), name="k-means++", data=data)
bench_k_means(KMeans(init='random', n_clusters=n_digits, n_init=10),
name="random", data=data)

# in this case the seeding of the centers is deterministic, hence
# we run the kmeans algorithm only once with n_init=1
pca = PCA(n_components=n_digits).fit(data)
bench_k_means(KMeans(init=pca.components_,      n_clusters=n_digits,
n_init=1),
              name="PCA-based",
              data=data)
print(82 * '_')
```

```
##############################################################
# Visualize the results on PCA-reduced data

reduced_data = PCA(n_components=2).fit_transform(data)
kmeans = KMeans(init='k-means++', n_clusters=n_digits, n_init=10)
kmeans.fit(reduced_data)

# Step size of the mesh. Decrease to increase the quality of the VQ.
h = .02     # point in the mesh [x_min, x_max]x[y_min, y_max].

# Plot the decision boundary. For that, we will assign a color to
each
x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:,
0].max() + 1
y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:,
1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
y_max, h))

# Obtain labels for each point in mesh. Use last trained model.
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1)
plt.clf()
plt.imshow(Z, interpolation='nearest',
           extent=(xx.min(), xx.max(), yy.min(), yy.max()),
           cmap=plt.cm.Paired,
           aspect='auto', origin='lower')

plt.plot(reduced_data[:, 0], reduced_data[:, 1], 'k.', markersize=2)
# Plot the centroids as a white X
centroids = kmeans.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1],
            marker='x', s=169, linewidths=3,
            color='w', zorder=10)
plt.title('K-means clustering on the digits dataset (PCA-reduced
data)\n'
          'Centroids are marked with white cross')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.show()
```

PARTIE 2 :

La 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.

La description de l'implémentation de la méthode des CAH se trouve dans :

http://scikitlearn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html#sklearn.cluster.AgglomerativeClustering

Exécution de l'algorithme **CAH** se fait par l'utilisation de package **sklearn.cluster.AgglomerativeClustering** avec la commande suivante :

```
class sklearn.cluster.AgglomerativeClustering(n_clusters=2, affinity='euclidean', connectivity=None, compute_full_tree='auto', linkage='ward')
```

- **n_clusters:** le nombre des cluster à trouver (2 par défaut).
- **affinity :** ("euclidean" par défaut) "euclidean", "l1", "l2", "manhattan", "cosine", 'precomputed', est la métrique utilisé pour calculer la stratégie (linkage).
- **connectivity:** définit pour chaque échantillon une matrice de connectivité entre des échantillons voisins suivant une structure de données.
- **linkage:** {"ward", "complete", "average"}, ("ward" par défaut)
- Définit une distance entre groupes d'individus (appelé stratégie d'agrégation). Stratégie du saut minimum ou single linkage (la distance entre parties est la plus petite distance entre éléments des deux parties). Stratégie du saut maximum ou du diamètre ou complete linkage (la distance entre parties est la plus grande distance entre éléments des deux parties). Méthode du saut Ward (en espace euclidien, on agrège de manière à avoir un gain minimum d'inertie intra-classe à chaque itération).

Travail à faire :

1. Effectuez un clustering du jeu de données en utilisant l'algorithme CAH avec le critère d'inertie de Ward.
2. Varier le paramètre **n_clusters.**

3. Modifier l'attribut **linkage :** "complete", "average".
4. Calculez le score des modèles construits à l'aide de la méthode de Silhouette.
5. Calculer le temps pour chaque stratégie.
6. Exécutez et expliquez le code qui suit.
7. A votre avis, quels sont les avantages et les inconvénients de la méthode CAH.

```
from time import time
import numpy as np
from scipy import ndimage
from matplotlib import pyplot as plt

from sklearn import manifold, datasets

digits = datasets.load_digits(n_class=10)
X  =  digits.data
y = digits.target
n_samples, n_features = X.shape

np.random.seed(0)

def nudge_images(X, y):
    # Having a larger dataset shows more clearly the behavior of the
    # methods, but we multiply the size of the dataset only by 2, as
the
    # cost of the hierarchical clustering methods are strongly
    # super-linear in n_samples
    shift = lambda x: ndimage.shift(x.reshape((8, 8)),
                                    .3 * np.random.normal(size=2),
                                    mode='constant',
                                    ).ravel()
    X = np.concatenate([X, np.apply_along_axis(shift, 1, X)])
    Y = np.concatenate([y, y], axis=0)
    return X, Y

X, y = nudge_images(X, y)
#----------------------------------------------------------------------------------------
# Visualize the clustering
def plot_clustering(X_red, X, labels, title=None):
    x_min, x_max = np.min(X_red, axis=0), np.max(X_red, axis=0)
    X_red = (X_red - x_min) / (x_max - x_min)

    plt.figure(figsize=(6, 4))
    for i  in  range(X_red.shape[0]):
        plt.text(X_red[i, 0], X_red[i, 1], str(y[i]),
                 color=plt.cm.spectral(labels[i] / 10.),
                 fontdict={'weight': 'bold', 'size': 9})

    plt.xticks([])
    plt.yticks([])
    if title is not None:
        plt.title(title, size=17)
    plt.axis('off')
```

```
    plt.tight_layout()

#----------------------------------------------------------------------
# 2D embedding of the digits dataset
print("Computing embedding")
X_red = manifold.SpectralEmbedding(n_components=2).fit_transform(X)
print("Done.")

from sklearn.cluster import AgglomerativeClustering

for linkage in ('ward', 'average', 'complete'):
    clustering = AgglomerativeClustering(linkage=linkage,
n_clusters=10)
    t0 = time()
    clustering.fit(X_red)
    print("%s : %.2fs" % (linkage, time() - t0))

    plot_clustering(X_red, X, clustering.labels_, "%s linkage" %
linkage)


plt.show()
```

PARTIE 3

Compléments

Le but de cette partie est de tester d'autres méthodes de clustering, une méthode proposée dans Scikit-learn et une autre méthode.

1. Appliquez la méthode de clustering DBSCAN sur le même jeu de données.
2. Implémentez un algorithme hiérarchique descendant basé sur l'utilisation de k-means.
3. Comme précédemment évaluez les performances obtenues pour les différentes méthodes (temps de calcul, qualité des partitions, forme des clusters, etc.) pour le jeu de données Digits.