TP 1. Clustering avec K-Means

On se propose de réaliser un algorithme de clustering. La méthode choisie est K-Means. On l'appliquera à deux bases de données Iris et Abalone.

Q.0 Les bases de données utilisées dans la suite de ce TP et des prochains se trouvent dans : https://github.com/renatopp/arff-datasets/ . Clônez ce dépôt git dans un dossier que vous nommerez data-mining et où vous stockerez également le présent fichier tp_clustering.ipynb.

Description de la base de données `Iris` Les Iris de différentes familles sont caractérisées par des longueurs et des largeurs de pétales et de sépales différentes. La base de données contient 150 instances. Les différents champs de la base sont : Sepal length, Sepal width, Petal length, Petal width

Description de la base de données `Abalone` Un abalone est un petit mollusque. On désire connaître son âge à partir de certaines caractéristiques physiques. En fait l'âge de l'animal peut se calculer en comptant le nombre d'anneaux et en y ajoutant 1,5. La base de données contient 4177 instances. Les différents champs de la base sont : Sex, Length, Diameter, Height, Whole weight, Shucked weight, Viscera weight, Shell weight, Class_Rings

Q.1 Grâce au préambule des bases de données iris.arff et abalone.arff, expliquer à quoi correpondent les différents champs de chacune des bases.

Q.2 Quel est le nombre de classes de chacune des bases ? Comment peut être utilisée cette donnée lors du clustering ?

Q.3 Donner le type de chacun des champs de chacune des bases. Puis, déterminer une distance (d_c) à utiliser pour chacun d'eux. Expliquer vos choix.

Q.4 Déterminer la distance (D) entre deux instances de la base :

- A quoi doit-on faire attention ?
- Donner la formulation choisie.

Dans la suite, nous allons implémenter la méthode K-Means en python. Nous commençons par importer les bibliothèques nécessaires.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
from scipy.io import arff
from scipy.spatial.distance import cdist, hamming
```

Ensuite, nous chargeons les deux bases de données.

```
def load_iris():
    data, meta = arff.loadarff('arff-datasets-master/classification/iris.arff')
    df = pd.DataFrame(data)
    df['class'] = df['class'].apply(lambda x: x.decode('utf-8'))  # Decode bytes to string
    class_mapping = {label: idx for idx, label in enumerate(df['class'].unique())}
    df['class'] = df['class'].map(class_mapping)
    print(df)
    return df.drop(columns=['class']).values, df['class'].values

def load_abalone():
    data, meta = arff.loadarff('arff-datasets-master/regression/abalone.arff')
    df = pd.DataFrame(data)
    df['Sex'] = df['Sex'].apply(lambda x: x.decode('utf-8'))
    sex_mapping = {'M': 0, 'F': 1, 'I': 2}
    df['Sex'] = df['Sex'].map(sex_mapping)
    print(df)
    return df.drop(columns=['Class_Rings']).values, df['Class_Rings'].values

abalone_X, abalone_y = load_abalone()
iris_X, iris_y = load_iris()
```

Nous avons maintenant des arrays numpy qui contiennent les variables "features" des bases de données abalone et iris.

```
print(abalone_X)
print(iris_X)
```

Nous voulons créer une fonction compute_distances qui prend comme variables

- un array numpy représentant un tableau de données,
- un array numpy représentant les centroïdes,
- une liste d'indices indiquant la position de variables catégoriques

et qui retourne un array numpy contenant une distance entre chacune des données et chacun des centroïdes, qui prend en compte la présence de variables catégoriques.

Q.5 S'aider de la structure suivante pour coder la fonction souhaitée.

```python
def distance_categorical(a,b):
    ... # calcule une distance entre deux vecteurs contenant uniquement des variables catégoriques

def distance_numerical(a,b):
    ... # calcule une distance entre deux vecteurs contenant uniquement des variables numériques

def compute_distance(X, centroids, categorical_idx=None):
    distances = np.zeros((X.shape[0], centroids.shape[0]))

    for i, sample in enumerate(X):
        samples_noncategorical_features = sample[np.setdiff1d(np.arange(len(sample)), categorical_idx)]
        for j, centroid in enumerate(centroids):
            centroid_noncategorical_features = centroid[np.setdiff1d(np.arange(len(sample)), categorical_idx)]
            categorical_dist = ...
            numerical_dist = ...
            distances[i, j] = ...   # Combiner les deux distances

    return distances
```

Nous voulons aussi créer une fonction distance_to_prev_centroids qui prend comme variables

- un array numpy représentant un tableau de centroïdes,
- un array numpy représentant les anciens centroïdes,
- une liste d'indices indiquant la position de variables catégoriques

et qui retourne un array numpy contenant une distance entre les centroïdes et les anciens centroïdes correspondants.

Q.6 S'aider de la structure suivante pour coder la fonction souhaitée.

```python
def distance_to_prev_centroids(prev_centroids, centroids, categorical_idx=None):
    distances = np.zeros(centroids.shape[0])
    for i, centroid in enumerate(centroids):
        prev_centroid = prev_centroids[i]
        centroid_noncategorical_features = centroid[np.setdiff1d(np.arange(len(sample)), categorical_idx)]
        prev_centroid_noncategorical_features = prev_centroid[np.setdiff1d(np.arange(len(sample)), categorical_idx)]
        categorical_dist = ...
        numerical_dist = ...
        distances[i] = ...   # Combiner les deux distances
```

Nous allons maintenant coder une fonctions kmeans qui exécute l'algorithme K-means.

Q.7 Expliquer à quoi correspondent les paramètres de cette fonction, et ce que retourne la fonction.

Q.8 Grâce à la structure suivante, coder la fonction kmeans.

```python
def kmeans(X, k, max_iters=100, tol=1e-4, categorical_idx=None):
    n_samples, n_features = X.shape
    centroids = X[random.sample(range(n_samples), k)]
    prev_centroids = np.zeros(centroids.shape) # prev_centroids tracke les anciens centroïdes pour pouvoir vérifier la convergence
    labels = np.zeros(n_samples)

    for _ in range(max_iters):
        distances = ... # tableau des distances de chacune des instances à chacun des centroïdes
        labels = ... # tableau contenant les identifiants des clusters auxquels chacune des instances est assignée

        # Calcul des nouveaux centroïdes
        for i in range(k):
            if np.any(labels == i):
                centroids[i] = ## calcul du nouveau centroïde du cluster associé au précédent centroïde i

        # Vérification de la convergence

        if distance_to_prev_centroids(centroids, prev_centroids, categorical_idx) < tol:
            break
        prev_centroids = centroids.copy()

    return labels, centroids
```

Q.9 Déterminer des mesures (et des moyens visuels) pour valider le clustering proposé par l'algorithme K-means.

Ici nous choisissons d'implémenter une fonction qui calcule le silhouette_score.

```python
def silhouette_score(X, labels):
    n = len(X)
```

```
    k = len(set(labels))
    silhouette_values = np.zeros(n)

    for i in range(n):
        same_cluster = X[labels == labels[i]]
        other_clusters = [X[labels == c] for c in set(labels) if c != labels[i]]

        a = np.mean([np.linalg.norm(X[i] - p) for p in same_cluster]) if len(same_cluster) > 1 else 0
        b = min([np.mean([np.linalg.norm(X[i] - p) for p in cluster]) for cluster in other_clusters]) if other_clusters else 0

        silhouette_values[i] = (b - a) / max(a, b)

    return np.mean(silhouette_values)
```

Nous souhaitons normaliser les données en utilisant une normalisation Min-Max.

Q.10 À quoi peut donc servir cette normalisation ? Implémenter la fonction normalize.

```
def normalize(X):
    ...
```

On normalise maintenant les jeux de données.

```
X_iris = normalize(X_iris)
X_abalone = normalize(X_abalone)
```

Puis on choisit d'exécuter K-Means pour iris (normalisé), puis pour abalone (normalisé).

Q.12 Compléter le code suivant.

```
y_pred_iris, centroids_iris = ...
print(f"Silhouette Score for Iris: {silhouette_score(X_iris, y_pred_iris)}")


y_pred_abalone, centroids_abalone = ...
print(f"Silhouette Score for Abalone: {silhouette_score(X_abalone, y_pred_abalone)}")
```

Q.11 S'il reste du temps, visualiser les clusters formés sur la bases iris, en traçant les points correspondants aux deux premières coordonnées, et en colorant les clusters de la même façon.