UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIENE

FACULTE D'INFORMATIQUE

Spécialité : Big Data Analytics

TIAD :

L'implémentation de l'algorithme de clustering améliorer

Réalisé par : BENYAMINA Ali

2022/2023

Introduction :

Le groupement (clustering) hiérarchique est une technique d'analyse exploratoire des données. C'est une technique non supervisée, qui consiste à construire un arbre de fusion binaire, en partant des éléments de données stockés aux feuilles (interprétés comme des ensembles singleton) et en fusionnant deux par deux les sous-ensembles « les plus proches » (stockés aux nœuds) jusqu'à atteindre la racine de l'arbre qui contient tous les éléments de X.

On note par D(Xi,Xj) la distance entre deux sous-ensembles de X.

Cette technique est aussi appelée clustering hiérarchique agglomératif, puisque nous partons des feuilles stockant des singletons (les x_i) et fusionnant itérativement les sous-ensembles jusqu'à atteindre la racine.

L'idée de l'algorithme de clustering hiérarchique agglomératif

Un tel clustering ne permet pas de connaître à l'avance le nombre de clusters. Le système prend en entrée l'ensemble de données et fournit en sortie une arborescence de clusters.

On compte deux classes d'algorithmes de ce type :

Clustering hiérarchique d'agglomérations (HAC) commence en bas, avec chaque donnée dans son propre cluster d'un singleton, et fusionne les clusters ensemble.

Le regroupement divisif commence avec toutes les données dans un grand cluster, puis les découpe jusqu'à chaque donnée est dans son propre cluster singleton.

Objectif de TP :

L'objectif de ce TP est d'implémenter l'algorithme de Clustering hiérarchique d'agglomérations c.-à-d. à partir d'une source de données on doit classifier ces données dans des clusters, avec une amélioration de critère de classification.

Ensuite avec les clusters crée on applique le principe de l'algorithme de K_means sans avoir donné le k.

A la fin on compare entre cette méthode avec l'algorithme K_means prédéfinie.

Critère de classification choisi :

Pour le critère on a calculé la moyenne des distances entre tous les points.

Pseudo Algorithme :

Entrée : data sets
Sortie : clusters

Début :

1. Placer le 1er objet dans un cluster
2. Calculer une liste des moyennes de distances entre chaque objet et le reste et choisir la moyenne minimum pour être notre critère.
3. Tant qu'il y a des objets à agglomérer
   3.1. Calculer une liste des distances entre les objets et les clusters existe (au début y a un seul cluster).
   3.2. Agglomérer les objets ayant la minimum distance entre elles et les groupes plus petits que le critère choisi.
   3.3. Calculer le centre du nouveau cluster
4. A partir des centres de ces clusters on effectue une nouvelle affectation des objets aux plus proches centres.
5. On s'arrête si aucun objet ne change de cluster
6. On calcule la distance inter-groupes et les distance intra de chaque cluster

Fait

Implémentation :

L'implémentation a été faite avec python3.10

Voici la source de donnée fichiers.csv qu'on a utilisé pour notre TP.

A	B	C	D	E
PLAYER,Mat.x,Inns.x,NO,Runs.x,HS,Avg.x,BF,SR.x				
AB de Villiers,12,11,2,480,90,53.33,275,174.54				
Quinton de Kock,8,8,0,201,53,25.12,162,124.07				
Rahul Tewatia,8,5,2,50,24,16.66,43,116.27				
Rahul Tripathi,12,12,3,226,80,25.11,167,135.32				
Rashid Khan,17,7,2,59,34,11.8,31,190.32				
Ravichandran Ashwin,14,9,1,102,45,12.75,71,143.66				
Ravindra Jadeja,16,10,5,89,27,17.8,74,120.27				
Rinku Singh,4,4,0,29,16,7.25,31,93.54				
Rishabh Pant,14,14,1,684,128,52.61,394,173.6				
Tim Southee,8,4,2,52,36,26,46,113.04				
Tom Curran,5,4,1,23,18,7.66,28,82.14				
Vijay Shankar,13,11,7,212,54,53,148,143.24				
Virat Kohli,14,14,3,530,92,48.18,381,139.1				
Washington Sundar,7,6,3,65,35,21.66,38,171.05				
Wriddhiman Saha,11,10,2,122,35,15.25,102,119.6				
Yusuf Pathan,15,13,4,260,45,28.88,200,130				
Yuvraj Singh,8,6,0,65,20,10.83,73,89.04				

Les bibliothèques utilisent :

```
import pandas as pd
from collections import defaultdict
from math import sqrt
```

Les fonctions principales utilisées

- « distance »

```
def distance(a, b):

    dimensions = 8
    _sum = 0
    for dimension in range(1,dimensions):
        difference_sq = (a[dimension] - b[dimension]) ** 2
        _sum += difference_sq
    return sqrt(_sum)
```

Cette fonction calcule la distance euclidienne entre doux points.

- « Moyen »

```
def Moyen(T):
    n=len(T)
    Moy=0
    for i in range(n):
            Moy=Moy+T[i]
    M=Moy/(len(T))
    return M
```

Cette fonction return la moyenne d'une List des nombres réels.

- « clustering »

```
def clustering(groups, min_dist, data, i, distance=[]):
    for g in range(len(groups)):
        somme = 0
        # jusqu'a la fin des nombre de colomne
        for j in range(len(list(data.iloc[i, :]))):
            # difference entre la 1er elt dans le cluster et les elt pas encore cluster
            diff = dif(groups[g][0][j], data.iloc[i, j])
            somme = somme+diff
        distance.append(somme)


    if min(distance)<= min_dist:
        groups[distance.index(min(distance))].append(
            list(data.iloc[i, :]))  # on ajoute just qui respect le criter

    else:
        group = []
        # else on le met dans un nouveau cluster
        group.append(list(data.iloc[i, :]))
        groups.append(group)

    return groups
```

Cette fonction on là donne une ancienne List des clusters et un point de positions i dans notre data et elle return une semi-List des clusters regrouper selon le « min_dist » qui est la moyenne des distances entre les points.

- « principal »

```
def principal(data):
    groups = [[list(data.iloc[0, :])]]
    moy=[]
    for line in range(0, len(data)):
        distanc =[]
        g=line+1
        while g < len(data):
            somme = 0
            for j in range(len(list(data.iloc[line, :]))):
                # difference entre la 1er elt dans le cluster et les elt pas encore cluster
                diff = dif(data.iloc[line, j], data.iloc[g, j])
                somme = somme+diff
            distanc.append(somme)
            g=g+1
        try:
          mindist= Moyen(distanc)
        except:
            continue
        moy.append(mindist)
    min_dist=min(moy)
    print(min_dist)
    for row in range(1, len(data)):
       groups = clustering(groups, min_dist, data, row,[])
    return groups
```

Cette fonction return un ensemble des clusters tel que chaque cluster, la distance entre ces points est inferieure a la moyenne des distances entre tous points.

- « center »

```
def center(grp):
    centers=[]
    new_center = []
    for j in range(1,len(list(data.iloc[1, :]))):
        dim_sum = 0
        for i in range(len(grp)):
            dim_sum +=grp[i][j]
        new_center.append(dim_sum / len(grp))
    new_center.insert(0,grp[0][0])
    return new_center
```

La fonction center calcule le centre de gravite d'un cluster.

- « assign_points »

```python
def assign_points(data_points, centers):
    """
    on la donnee un ensemble de données et une liste de points entre les autres points,
    assigner chaque point à un index qui correspond à l'index
    du point central sur sa proximité à ce point.
    Retourner un tableau d'index de centres qui correspondent à
    un index dans l'ensemble de données,
    """

    assignments = []
    for point in data_points:
        shortest = float('inf')  # positive infinity
        shortest_index = 0
        for i in range(len(centers)):
            val = distance(point, centers[i])
            if val < shortest:
                shortest = val
                shortest_index = i
        assignments.append(shortest_index)
    return assignments
```

Cette fonction return un tableau d'index de centres qui correspondent à un index dans l'ensemble de données.

- « update_centers »

```python
def update_centers(data_set, assignments):
    """
    Accepte un ensemble de données et une liste de tâches; les index
    des deux listes correspondent l'une à l'autre.
    Calculer le centre pour chacun des clusters assignés.
    Retourner des nouveux centres.
    """
    new_means = defaultdict(list)
    centers = []
    for assignment, point in zip(assignments, data_set):
        new_means[assignment].append(point)
        #print(f'assignement{assignment}\n point{point}')
    for points in new_means.values():
        centers.append(center(points))

    return centers
```

La fonction update_centers calcule et return le centre de gravité de nouveau clusters.

- « k_means »

```python
def k_means(dataset, centers):
    #print(f'data\n{dataset} \n centers{centers}')
    assignments = assign_points(dataset,centers)
    old_assignments = None
    itr=1
    while assignments != old_assignments:
        print(f'--------- iteration :{itr} ---------')
        new_centers = update_centers(dataset, assignments)
        old_assignments = assignments
        assignments = assign_points(dataset, new_centers)
        itr+=1
    return assignments, dataset
```

Dans cette fonction (en utilisant les fonctions précédant) return un tableau d'index de centres qui correspondent à un index dans l'ensemble de données.

Ps. A chaque fois on a une nouvelle affectation dans cette fonction on print itération :{nbr_iteration}

```
p = 1
centers=[]
groups = principal(data)
for g in groups:
    print(f'\n cluster  {p}')
    for i in g:
        print([''.join(str(i))])
    print(f'\n centre de gravity de cluster {p}')
    new_center=center(g)
    centers.append(new_center)
    print(new_center)
    p = p+1
print('\n.............................affectation de chaque données au cluster le plus proche K-Means.
a,b=k_means(data1,centers)
for i in range(len(a)):
    print(f'cluster{a[i]} : {b[i]}')
```

A la fin on fait appel a notre doux main fonctions « principal » et « k_means » pour voir le résultat dans le Terminal.

Voici le résultat :

D'abord on donne le chemin de notre source de données.

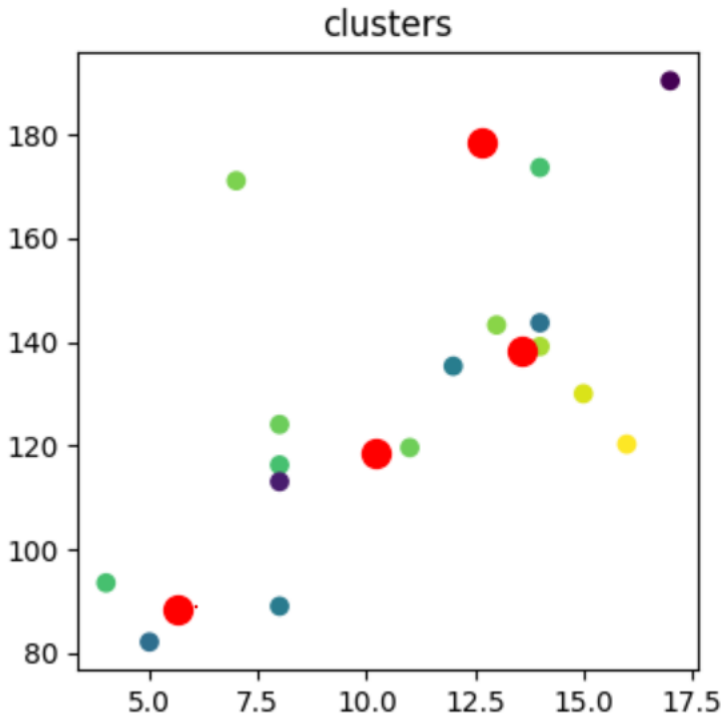
```
enter file link : data.csv
```

```
 cluster  1
["['AB de Villiers', 12, 11, 2, 480, 90, 53.33, 275, 174.54]"]
["['Virat Kohli', 14, 14, 3, 530, 92, 48.18, 381, 139.1]"]

 centre de gravity de cluster 1
['AB de Villiers', 13.0, 12.5, 2.5, 505.0, 91.0, 50.754999999999995, 328.0, 156.82]

 cluster  2
["['Quinton de Kock', 8, 8, 0, 201, 53, 25.12, 162, 124.07]"]
["['Rahul Tripathi', 12, 12, 3, 226, 80, 25.11, 167, 135.32]"]
["['Vijay Shankar', 13, 11, 7, 212, 54, 53.0, 148, 143.24]"]
["['Yusuf Pathan', 15, 13, 4, 260, 45, 28.88, 200, 130.0]"]

 centre de gravity de cluster 2
['Quinton de Kock', 12.0, 11.0, 3.5, 224.75, 58.0, 33.0275, 169.25, 133.1575]
```

```
 cluster  3
["['Rahul Tewatia', 8, 5, 2, 50, 24, 16.66, 43, 116.27]"]
["['Rashid Khan', 17, 7, 2, 59, 34, 11.8, 31, 190.32]"]
["['Ravichandran Ashwin', 14, 9, 1, 102, 45, 12.75, 71, 143.66]"]
["['Ravindra Jadeja', 16, 10, 5, 89, 27, 17.8, 74, 120.27]"]
["['Rinku Singh', 4, 4, 0, 29, 16, 7.25, 31, 93.54]"]
["['Tim Southee', 8, 4, 2, 52, 36, 26.0, 46, 113.04]"]
["['Tom Curran', 5, 4, 1, 23, 18, 7.66, 28, 82.14]"]
["['Washington Sundar', 7, 6, 3, 65, 35, 21.66, 38, 171.05]"]
["['Wriddhiman Saha', 11, 10, 2, 122, 35, 15.25, 102, 119.6]"]
["['Yuvraj Singh', 8, 6, 0, 65, 20, 10.83, 73, 89.04]"]

 centre de gravity de cluster 3
['Rahul Tewatia', 9.8, 6.5, 1.8, 65.6, 29.0, 14.766, 53.7, 123.89299999999999]

 cluster  4
["['Rishabh Pant', 14, 14, 1, 684, 128, 52.61, 394, 173.6]"]

 centre de gravity de cluster 4
['Rishabh Pant', 14.0, 14.0, 1.0, 684.0, 128.0, 52.61, 394.0, 173.6]
```

```
.............................affectation de chaque données au cluster le plus proche K-Means.............................

--------- iteration :1 ---------
--------- iteration :2 ---------
[0, 1, 0, 1, 1, 1, 1, 2, 1, 1, 0, 3, 1, 1, 0, 1]
cluster0 : ['Quinton de Kock', 8, 8, 0, 201, 53, 25.12, 162, 124.07]
cluster1 : ['Rahul Tewatia', 8, 5, 2, 50, 24, 16.66, 43, 116.27]
cluster0 : ['Rahul Tripathi', 12, 12, 3, 226, 80, 25.11, 167, 135.32]
cluster1 : ['Rashid Khan', 17, 7, 2, 59, 34, 11.8, 31, 190.32]
cluster1 : ['Ravichandran Ashwin', 14, 9, 1, 102, 45, 12.75, 71, 143.66]
cluster1 : ['Ravindra Jadeja', 16, 10, 5, 89, 27, 17.8, 74, 120.27]
cluster1 : ['Rinku Singh', 4, 4, 0, 29, 16, 7.25, 31, 93.54]
cluster2 : ['Rishabh Pant', 14, 14, 1, 684, 128, 52.61, 394, 173.6]
cluster1 : ['Tim Southee', 8, 4, 2, 52, 36, 26, 46, 113.04]
cluster1 : ['Tom Curran', 5, 4, 1, 23, 18, 7.66, 28, 82.14]
cluster0 : ['Vijay Shankar', 13, 11, 7, 212, 54, 53, 148, 143.24]
cluster3 : ['Virat Kohli', 14, 14, 3, 530, 92, 48.18, 381, 139.1]
cluster1 : ['Washington Sundar', 7, 6, 3, 65, 35, 21.66, 38, 171.05]
cluster1 : ['Wriddhiman Saha', 11, 10, 2, 122, 35, 15.25, 102, 119.6]
cluster0 : ['Yusuf Pathan', 15, 13, 4, 260, 45, 28.88, 200, 130]
cluster1 : ['Yuvraj Singh', 8, 6, 0, 65, 20, 10.83, 73, 89.04]
PS C:\Users\BIGNETWORK\Desktop\python>
```

Pour voir bien les résultats on a ploté les clusters ainsi que son centre de gravite voici le résultat :



La comparaison entre notre méthode et la fonction prédéfinie on python

- Les indexes de centres qui correspondent à un index dans l'ensemble de données de notre code :

```
[0, 1, 0, 1, 1, 1, 1, 2, 1, 1, 0, 3, 1, 1, 0, 1]
```

- Les indexes de centres qui correspondent à un index dans l'ensemble de données de K_means prédéfinie :

```
X1 = pd.read_csv('data.csv')
kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300, n_init=10, random_state=0).fit(X1)
kmeans.labels_
print(kmeans.labels_)
```

```
[3 3 2 0 2 3 1 0 3 1 2 2 0 3 2 1]
```

La comparaison par rapport à la distance inter-groupes et les distances intra de chaque groupe entre les deux méthodes :

- La distance inter-groupes et les distances intra de chaque groupe de notre code :

```
la moyenne des distances inter-groupe : 388.39477751943946

la distance intra de cluster 1 : 117.39047022650519

la distance intra de cluster 2 : 73.17028986732278

la distance intra de cluster 3 : 61.41133923482784

la distance intra de cluster 4 : 61.41133923482784
```

- La distance inter-groupes et les distances intra de chaque groupe de K_means prédéfinie :

```
la moyenne des distances inter-groupe : 179.11348159412742

la distance intra de cluster 1 : 547.1551110477624

la distance intra de cluster 2 : 296.8868666921712

la distance intra de cluster 3 : 275.5230029926235

la distance intra de cluster 4 : 212.7286842376933
```

La meilleure méthode c'est la méthode qui donne une distance inter-groupes maximal et des distances intra minimum

En remarque que notre méthode donne la meilleure classification par rapport à l'algorithme de K_means prédéfinie.

Conclusion :

Ces algorithmes sont des algorithmes d'apprentissage automatique non supervisé qui fait partie d'un ensemble de techniques et d'opérations de données très approfondi dans le domaine de la Data science. Ce sont les algorithmes les plus rapide et les plus efficace pour catégoriser les points de données en groupes, même lorsque très peu d'information est disponible sur les données.

De plus, comme pour tout autre apprentissage non supervisé, il est nécessaire de comprendre les données avant d'adopter la technique qui convient le mieux à un ensemble de données donné pour résoudre les problèmes. Considérer le bon algorithme, en retour, peut faire gagner du temps et des efforts et aider à obtenir des résultats plus précis.