CHAPITRE

I Clustering

1 Introduction

2 Algorithme K-Means

Voyons comment fonctionne le clustering k-means avec un peu de pratique en même temps. Tout d'abord, intéressons nous à la fonction **make_blobs** de la librairie sklearn. Cette fonction permet de générer aléatoirement des jeux de données. Nous allons créer quatre clusters aléatoires en utilisant **make_blobs** pour nous aider dans notre tâche.

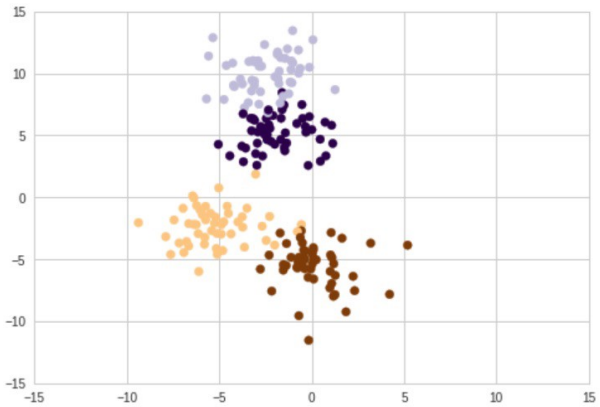
```
# import statements
from sklearn.datasets import make_blobs
import numpy as np
import matplotlib.pyplot as plt

# create blobs
data = make_blobs(n_samples=200, n_features=2, centers=4, cluster_std=1.6, random_state=50)

# create np array for data points
points = data[0]
classes = data[1]

# create scatter plot
plt.scatter(points [:,0], points [:,1], c=classes, cmap='viridis')
plt.xlim(-15,15)
plt.ylim(-15,15)
```

Voici à quoi doivent ressembler vos données. Nous avons quatre groupes de couleurs (une couleur = une classe supposée), mais il y a un certain chevauchement avec les deux groupes du haut, ainsi que les deux groupes du bas. C'est ce type de situation qui rend le clustering délicat.

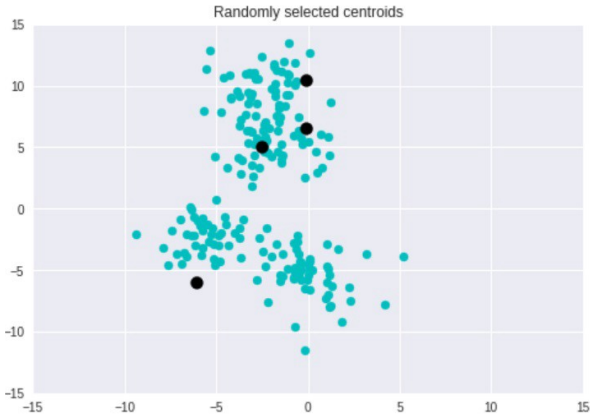


Remarque 2 - Pas de couleurs

En réalité, pour le clustering, il n'y a pas de couleur connue initialement. Dans la visualisation ci-dessus, on l'illustre pour voir les données. C'est à l'algorithme de clustering d'identifier les "bonnes" classes.

2.1 Principe de l'algorithme

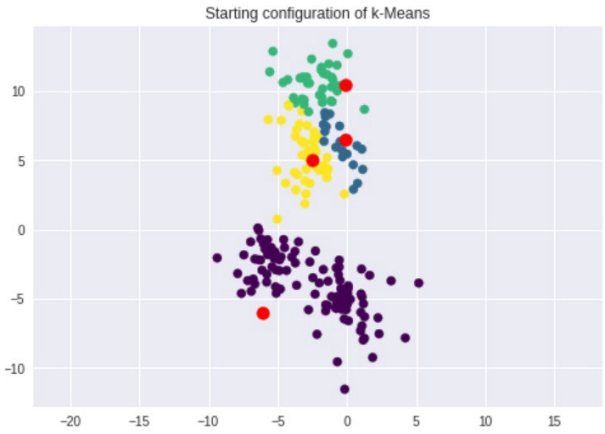
Passons à la description du k-means. La première étape de la classification en k-means consiste à sélectionner des centroïdes aléatoires. Puisque notre $k = 4$ dans ce cas, nous aurons besoin de 4 centroïdes aléatoires. Voici à quoi cela ressemblait dans mon implémentation à partir de zéro.



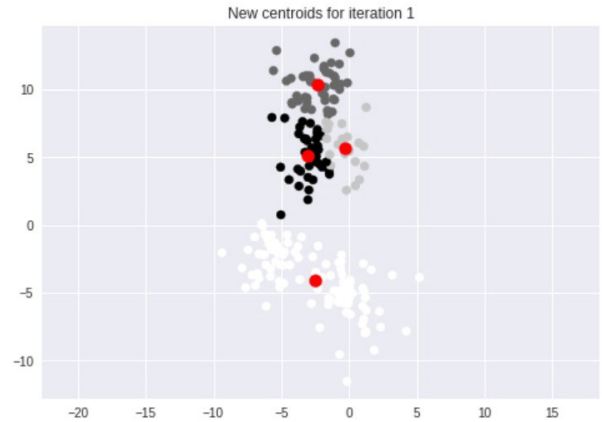
Ensuite, nous prenons chaque point et trouvons le centroïde le plus proche. Il existe différentes façons de mesurer la distance, ici c'est la distance euclidienne, qui peut être mesurée à l'aide de `np.linalg.norm` en Python. Attention néanmoins de ne pas prendre n'importe quoi comme mesure de dissimilarité. Pour l'algorithme k-means, seule l'utilisation d'une vraie distance (c'est-à-dire, une mesure défini, positive et respectant l'inégalité triangulaire) guarantie la converge de l'algorithme.

Remarque 3 - Importance de la distance

Lorsqu'on utilise un algorithme de clustering, le choix de la distance, ou de la mesure de similarité, impacte très fortement le résultat. La distance contient la connaissance de l'expert sur son jeu de données ... elle dit comment deux exemples doivent être considérés comme similaire ou non. La méthode de clustering se contente ensuite de faire un arrangement optimal selon se critère.



Maintenant que nous avons 4 nouveaux clusters, nous trouvons les nouveaux centroïdes des clusters. De nouveau, c'est la distance euclidienne qui est utilisée pour calculer un centroïde.



Ensuite, nous associons chaque point au centroïde le plus proche, en répétant le processus, jusqu'à ce que nous ne puissions plus améliorer les clusters.

Les trois points importants pour la méthode de clustering sont :
- le choix de la distance
- le nombre de cluster
- la méthode d'initialisation des centroïdes

2.2 Test du k-means

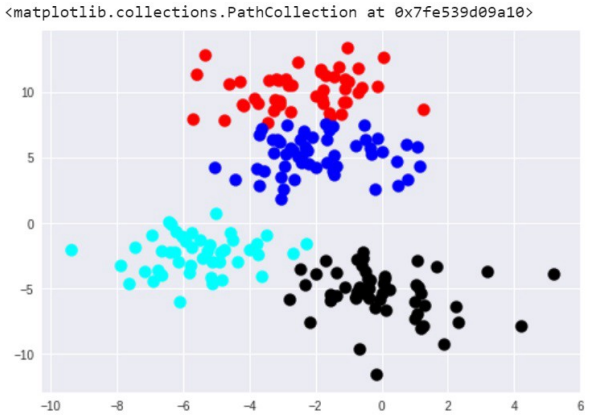
On effectue maintenant la classification à l'aide des sklearn en utilisant le script ci-dessous.

```
# import KMeans
from sklearn.cluster import KMeans

# create kmeans object
kmeans = KMeans(n_clusters=4)# fit kmeans object to data
kmeans.fit(points)# print location of clusters learned by kmeans object
print(kmeans.cluster_centers_)# save new clusters for chart
y_km = kmeans.fit_predict(points)

plt.scatter(points[y_km ==0,0], points[y_km == 0,1], s=100, c='red')
plt.scatter(points[y_km ==1,0], points[y_km == 1,1], s=100, c='black')
plt.scatter(points[y_km ==2,0], points[y_km == 2,1], s=100, c='blue')
plt.scatter(points[y_km ==3,0], points[y_km == 3,1], s=100, c='cyan')
```

Le résultat obtenu est donné par la figure ci-dessous. On constate que le résultat est ici assez proche de l'attendu.



Bon soit ... ca ressemble, mais comme on a des classes à ce jeu de données, on peut calculer une similarité entre la classification qui était attendue et cette que l'on a obtenu. Pour cela, j'utilise ici l'indice de rang. Cette mesure calcule la similarité entre deux clustering en prenant en compte toutes les paires d'échantillons et en comptant les paires attribuées dans le même groupe ou dans des groupes différents dans les groupes prédits et réels. Une autre mesure usuelle est le score de silhouette.

```
from sklearn import metrics
metrics.adjusted_rand_score(kmeans.labels_, classes)
metrics.metrics.silhouette_score(kmeans.labels_, classes)
```

2.3 Choix du nombre de cluster

Le script ci-dessous fait varier le nombre de clusters k et calcule un graphique de la qualité du clustering en fonction de k. On pourra noter que les paramètres du *blobs* ont ici changer : en ne fixant pas le paramètre **random_state** les données sont différentes à chaque exécution.

```
# create blobs
data = make_blobs(n_samples=200, n_features=2, centers=4, cluster_std=1.6)
# create np array for data points
points = data[0]
classes = data[1]

ks=range(2,8)
sims=[]
for k in ks:
  kmeans = KMeans(n_clusters=k)# fit kmeans object to data
  kmeans.fit(points)# print location of clusters learned by kmeans object
```

```
  sim=metrics.adjusted_rand_score(kmeans.labels_, classes)
  sims.append(sim)

plt.plot(ks, sims)
```

Analysez ce programme, exécutez le plusieurs fois et commentez ce que vous constatez.

Exercice 1 (Robustesse)

Question a) En reprenant le script de visualisation des données du *blobs*, visualiser les données générées pour un paramètre `cluster_std` qui vaut 1, 3, 10. En déduire l'effet de ce paramètre.

Question b) En vous inspirant du programme précédent, réaliser un programme qui affiche une courbe représentant la qualité du clustering en fonction du paramètre `cluster_std` entre 1 et 10 (de 1 en 1).

Question c) Analysez ensuite cette courbe.

3 Autres algorithmes de clustering

K-means est la forme de clustering la plus fréquemment utilisée en raison de sa rapidité et de sa simplicité. Une autre méthode de classification très courante est la classification hiérarchique, mais vous retrouver également fréquemment l'algorithme DBSCAN pour les images satellite. sklearn contient également ces classifieurs et également d'autres.

3.1 Agglomerative Hierarchical Clustering – CAH

La classification hiérarchique par agglomération diffère des k-moyennes. Plutôt que de choisir un certain nombre de clusters et de commencer par les centroïdes aléatoires, nous commençons plutôt avec chaque point de notre jeu de données en tant que "clusters" élémentaire. Nous trouvons ensuite les deux points les plus proches et les combinons en un nouveau cluster. Nous répétons le processus jusqu'à ce que nous ayons un seul cluster géant.
On obtient ainsi une structure qu'on appelle un dendogramme et qui représente de manière hiérarchique. Le script ci-dessous va permettre de réaliser un clustering des données du *blobs* selon la méthode agglomérative.

```
# import hierarchical clustering libraries
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering

# create dendogramme
dendrogram = sch.dendrogram(sch.linkage(points, method='ward'))
```

Pour réaliser un clustering, il suffit soit de choisir un nombre de cluster, soit de fixer un seuil de similarité pour "couper" l'arbre en autant de branche indépendantes qu'il y a de cluster sous la coupure.

```
# create clusters
hc = AgglomerativeClustering(n_clusters=4, affinity = 'euclidean', linkage = 'ward')
```

On peut ainsi visualiser les cluster de manière similaire aux k-means.

```
# save clusters for chart
y_hc = hc.fit_predict(points)

plt.scatter(points[y_hc ==0,0], points[y_hc == 0,1], s=100, c='red')
plt.scatter(points[y_hc==1,0], points[y_hc == 1,1], s=100, c='black')
plt.scatter(points[y_hc ==2,0], points[y_hc == 2,1], s=100, c='blue')
plt.scatter(points[y_hc ==3,0], points[y_hc == 3,1], s=100, c='cyan')
```

Cette méthode permet de ne pas fixer initialement le nombre de clusters à constituer, de plus, la contrainte sur la mesure de similarité est moins forte. Les paramètres principaux de cette méthodes sont :

- le dessimilarité (hyper-métrique)
- le seuil de coupure
- la stratégie d'agglomération
  - Ward minimizes the sum of squared differences within all clusters. It is a variance-minimizing approach and in this sense is similar to the k-means objective function but tackled with an agglomerative hierarchical approach.

— Maximum or complete linkage minimizes the maximum distance between observations of pairs of clusters.
— Average linkage minimizes the average of the distances between all observations of pairs of clusters.
— Single linkage minimizes the distance between the closest observations of pairs of clusters.

Exercice 2 (Effet de la distance) *En utilisant le* **linkage**=*"average", faites varier la dissimilarité (* **affinity** *) en utilisant les valeurs de paramètre "cosine", "euclidean" ou "cityblock" et observer les différences.*
Vous pourrez faire en sorte de réaliser un programme qui effectue une boucle pour créer tous les graphique d'un seul coup.

4 Application au clustering d'une image satellite

4.1 Classification d'une image

```
import gdal
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from random import randint


dataset = gdal.Open("Senegal_Diourbel_NDVI_2002.tif", 0 )

bands=[]
for i in range(dataset.RasterCount):
    band=dataset.GetRasterBand(i+1)
    bands.append( band.ReadAsArray() )
data=np.array(bands)
#nettoyage des variables inutiles )
del(bands,band,i)

print (np.shape(data)) #output: (23, 261, 499)

## Affichage d'une bande de l'image ( ici , la 13eme date)
plt.figure()
plt.imshow(data[12,:,:])

## transformation des 23 couches de matrice 2D, en matrice 2D de pixels de dimension 23

data = np.dstack(data)

print (np.shape(data)) #output: (261, 499, 23)

## Affichage d'une composition colorée à partir des bandes 0 à 3 de l'image
plt.figure()
plt.imshow(data[:,:,0:3])

## transformation en vecteurs de séries temporelles
tsdata = data.reshape( (data.shape[0]*data.shape[1],data.shape[2]) )

print (np.shape(tsdata)) #output: (130239, 23)
## Affichage d'une série temporelle du jeu de données
plt.figure()
plt.plot(tsdata[230])

########################## TIME SERIES CLUSTERING ################

clusterer = KMeans(n_clusters=10)

trainsize =1000

trainset = tsdata[ [randint(0,tsdata.shape[0]) for i in range( trainsize )] ]

clusters = clusterer.fit( trainset )
clusters = clusterer.predict(tsdata)

########################## image output ################
```

```
clusters = clusters.reshape( (data.shape[0], data.shape[1]) )

## Affichage de l'image résultant du clustering
plt.figure()
plt.imshow(clusters)

## enregistrement de l'image
driver = gdal.GetDriverByName( "GTiff" )
dst_ds = driver.Create( "output.tif", data.shape[1], data.shape[0], 1, gdal.GDT_Int32 )
dst_ds.SetProjection( dataset.GetProjection() )
dst_ds.SetGeoTransform( dataset.GetGeoTransform() )
dst_ds.GetRasterBand(1).WriteArray( clusters )
dst_ds=None
```

Exercice 3 (Matrice de confusion) *On cherche à comparer deux résultats clustering successifs pour voir si la méthode est stable ou non. Pour cela on souhaite utiliser une matrice de confusion.*

Question a) Commencer par consulter la documentation en ligne sur les matrices de confusions : `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html`

Question b) Exécuter un premier clustering et sauver le vecteur résultant du clustering dans des variables **clusters1** et **clusters2**

Question c) Calculer la matrice de confusion avec Python et analysez le résultat

Question d) Réaliser la même opération en comparant des résultats de deux clustering obtenus avec des différents nombres de classes

Exercice 4 (Kappa en fonction du nombre d'exemples) *On cherche de nouveau à comparer deux résultats de clustering, on souhaite utiliser l'indice de Kappa qui résume la matrice de confusion en une valeur.*

Question a) Commencer par consulter la documentation en ligne sur l'indice de Kappa : `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.cohen_kappa_score.html`

Question b) Faire un programme qui calcule deux clustering avec des échantillons différents et calcul le Kappa

Question c) Modifier le programme pour obtenir une liste de valeurs de Kappa pour des tailles d'échantillon allant de 100 à 100000 en échelle logarithmique.
Vous utiliserez la commande **np.logspace***(2.0, 6.0,* **num***=5) pour générer les tailles d'échantillon.*

Question d) Afficher la courbe de l'évolution du Kappa