DÉPARTEMENT MATHÉMATIQUES ET INFORMATIQUE

Rapport du Projet Data Mining , Machines et Deep Learning

Filière :
« Ingénierie Informatique : Big Data et Cloud Computing »

II-BDCC

Utiliser la méthode K-means pour un problème de prévision

Réalisé par :
Tarik Fertahi
Amina Dahmouni
Zakaria Mansouri

Encadré par :
M. Zakaria EN-NAIMANI

Année Universitaire : 2022-2023

Introduction

Machine Learning (apprentissage automatique) est un sous-domaine de l'Intelligence Artificielle. C'est un ensemble de méthodes et de techniques ou algorithmes qui permettent aux ordinateurs de construire des modèles et de générer des prédictions sans utiliser d'instructions explicites ou être guidés par un humain.

Types d'apprentissage

Il existe 3 types d'apprentissage sont les suivantes :

1- Apprentissage supervisé

L'apprentissage supervisé (supervised learning en anglais) en input des individus labellisés, et l'objectif est de classer un nouvel individu non labellisé. C'est une tâche d'apprentissage automatique consistant à apprendre une fonction de prédiction à partir d'exemples annotés, au contraire de l'apprentissage non supervisé. On distingue les problèmes de régression des problèmes de classement. Ainsi, on considère que les problèmes de prédiction d'une variable quantitative sont des problèmes de régression tandis que les problèmes de prédiction d'une variable qualitative sont des problèmes de classification.

2- Apprentissage non supervisé

L'apprentissage non supervisé en input des individus non labellisés, et l'objectif est de faire le regroupement. Désigne la situation d'apprentissage automatique où les données ne sont pas étiquetées (par exemple étiquetées comme « balle » ou « poisson »). Il s'agit donc de découvrir les structures sous-jacentes à ces données non étiquetées. Puisque les données ne sont pas étiquetées, il est impossible à l'algorithme de calculer de façon certaine un score de réussite. Ainsi, les méthodes non supervisées présentent une auto-organisation qui capture les modèles comme des densités de probabilité ou, dans le cas des réseaux de neurones, comme combinaison de préférences de caractéristiques neuronales encodées dans les poids et les activations de la machine.

3- Apprentissage par renforcement

L'apprentissage par renforcement ou Reinforcement Learning est une méthode de Machine Learning. Elle consiste à entraîner des modèles d'intelligence artificielle d'une manière bien spécifique. La machine apprend de ses propres erreurs. La machine fait des choix, ensuite récompensée s'il s'agit d'un bon choix, si non pénalisé.

Techniques descriptives du Datamining

Les techniques descriptives visent à mettre en évidence des informations présentes mais cachées par le volume des données. Ces techniques consistent à réduire, résumer, synthétiser les données.

Méthodes descriptives

Il existe 3 méthodes descriptives sont les suivantes :
1- Analyse factorielle
2- Détection d'associations entre les individus
3- Segmentation

Segmentation

Segmentation c'est une méthode descriptive qui permet de regrouper des objets en groupes, ou classes, ou familles, ou segments, ou clusters, de sorte que :

- 2 objets d'un même groupe se ressemblent le plus possible,
- 2 objets de groupes distincts diffèrent le plus possible.

k-means

k-means est un outil de segmentation qui permet de répartir un ensemble de données en classes homogènes.

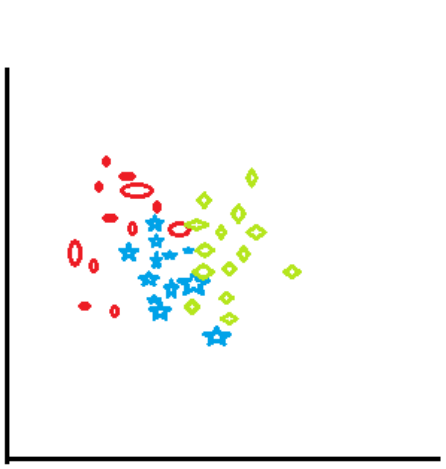


fig 1: before applying k-means clustering

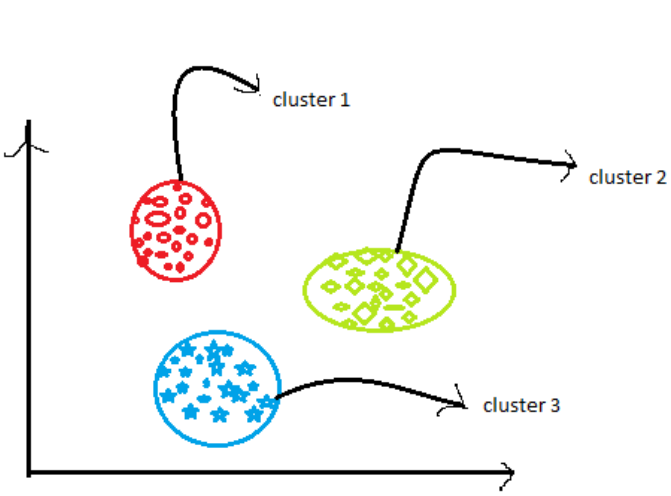


fig 2: After applying K-means clustering

Problématique

Utiliser l'algorithme Kmeans pour prédire le type d'une Iris flower.

Le jeu de données :

Le jeu de données Iris a été initialement publié à l' UCI Machine Learning Repository: Iris Data Set. Ce *Data Set* de 1936 est souvent utilisé pour tester des algorithmes d'apprentissage automatique et des visualisations.

Le jeu de données *Iris* contient trois variantes de la fleur *Iris.* Il contient 150 instances (ligne du jeu de données). Chaque instance est composée de quatre attributs pour décrire une fleur d'Iris. Le jeu de données est étiqueté par le type de fleur. Ainsi pour quatre attributs décrivant une fleur d'Iris, on saura de quelle variante il s'agit.

Finalement, le jeu de données ne contient pas de valeurs manquantes. Ce qui nous dispense de les traiter.

Iris Versicolor **Iris Setosa** **Iris Virginica**

Implémentation de K-Means:

Import des librairies nécessaires

```
#mathematical operations on arrays
import numpy as np
#to analyze data
import pandas as pd
import matplotlib.pyplot as plt
#Seaborn to visualize the data and make it more and more undertakable by the user
import seaborn as sns
#Le package sklearn datasets intègre quelques petits ensembles de données
from sklearn import datasets
from sklearn import cluster
from sklearn.cluster import KMeans
import random
import disEuc
```

Chargement des données

La librairie Sickit Learn offre des méthodes utilitaires pour charger des jeux de données populaires comme celui d'Iris. Ces méthodes se retrouvent dans la classe **datasets.**

Pour charger notre jeu de données Iris, on utilise la méthode **load_iris()**.

```
# Import the Iris flower dataset
iris = datasets.load_iris()
print("Dataset loaded successfully")
```

```
Dataset loaded successfully
```

Utilisation de la librairie Pandas

Pandas est une librairie assez utilisée quand on fait du Machine Learning avec Python. Grâce à sa classe **DataFrame**, on peut manipuler aisément les données en format tabulaire.

```
#Creating data frame
Data = pd.DataFrame(iris.data, columns = iris.feature_names)
#Top values of Dataset
Data.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Informations générales sur notre dataset :

```
Data.shape
```

```
(150, 4)
```

Nous avons 150 échantillons et 4 caractéristiques.

```
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
dtypes: float64(4)
memory usage: 4.8 KB
```

Le jeu de données ne contient pas de valeurs manquantes.

Construction du modèle K-means

Choisissez le nombre de clusters k :

Nous avons utilisé la méthode Elbow pour déterminer le nombre optimal de clusters pour le clustering k-means.

C'est une étape fondamentale pour tout algorithme non supervisé qui consiste à déterminer le nombre optimal de clusters dans lesquels les données peuvent être regroupées.

Nous démontrons maintenant la méthode donnée en utilisant la technique de clustering K-Means en utilisant la bibliothèque Sklearn de python.

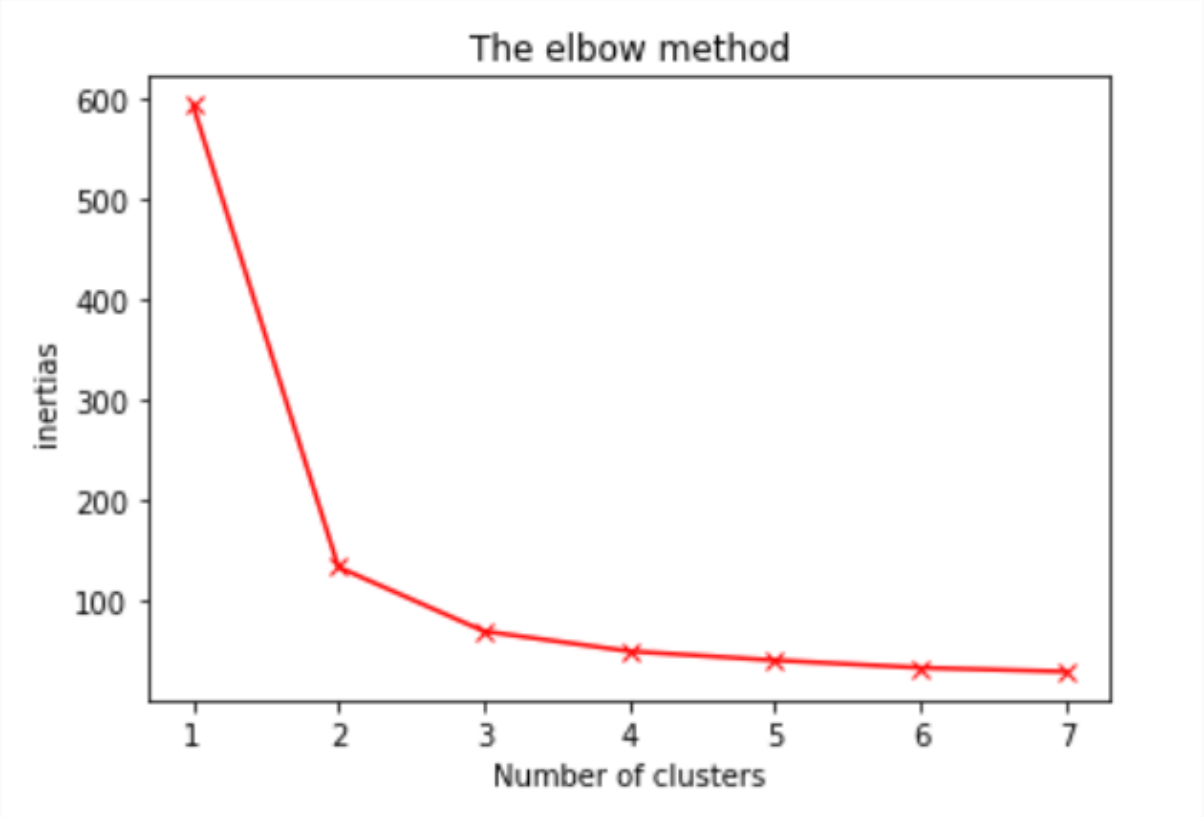
Pour trouver Inertie : C'est la somme des distances au carré des échantillons à leur centre de grappe le plus proche.

Nous itérons les valeurs de k de 1 à 8 et calculons l'inertie pour chaque valeur de k dans la plage donnée.

```
#Trouver des nombres de clusters pour Kmeans
x=Data.iloc[:,0:3].values
inertias=[]

# Trouver l'inertie sur diverses valeurs de k
for i in range(1,8):
    kmeans=KMeans(n_clusters = i, init = 'k-means++', max_iter = 100, n_init = 10, random_state = 0).fit(x)
    inertias.append(kmeans.inertia_)

plt.plot(range(1, 8), css, 'bx-', color='red')
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('inertias')
plt.show()
```



Pour déterminer le nombre optimal de clusters, nous devons sélectionner la valeur de k au « Elbow», c'est-à-dire le point après lequel l'inertie commence à diminuer de manière linéaire. Ainsi, pour Iris dataset, nous concluons que le nombre optimal de clusters pour les données est 3.

```
#Application du classificateur KMeans
train_data = np.array(iris.data)
```

```
clf = KMeansClass(k=3, max_iterations=200)
```

Définition de la classe KMeansClass avec le constructeur **__init__()** qui sera appelée lors de la création de l'objet pour l'initialisation:

```python
class KMeansClass():
    def __init__(self, k=3, max_iterations=500):
        self.k = k
        self.max_iterations = max_iterations
        self.kmeans_centroids = []
```

Initialiser les centroïdes aléatoirement :

```python
# Initialiser les centroïdes en tant qu'échantillons aléatoires
def _init_random_centroids(self, data):
    #samples=échantillons   features=caractéristiques
    n_samples, n_features = np.shape(data)
    #Python numpy. zeros() function returns a new array of given shape and type
    centroids = np.zeros((self.k, n_features))
    for i in range(self.k):
        centroid = data[np.random.choice(range(n_samples))]
        centroids[i] = centroid
    return centroids
```

Calculer la distance euclidien entre deux points :

```python
# Calculer la distance entre deux points
def euclidean_distance(pnt1, pnt2):
    if(len(pnt1) != len(pnt2)):
        raise Exception("Les deux points n'ont pas la même longueur")

    distance = 0
    for i in range(len(pnt1)):
        distance += pow((pnt1[i] - pnt2[i]), 2)

    return np.sqrt(distance)
```

Envoyer l'indice du centroïde le plus proche de l’échantillon :

```python
# Renvoie l'indice du centroïde le plus proche de l'échantillon
def _closest_centroid(self, sample, centroids):
    closest_i = None
    #inicialiser closest_distance par valeur supérieure illimitée pour la comparaison
    closest_distance = float("inf")
    for i, centroid in enumerate(centroids):
        distance =  self.euclidean_distance(sample, centroid)
        if distance < closest_distance:
            closest_i = i
            closest_distance = distance
    return closest_i
```

Affecter les échantillons aux centroïdes les plus proches pour créer des clusters :

```python
# Attribuez les échantillons aux centroïdes les plus proches pour créer des clusters
def _create_clusters(self, centroids, data):
    #n_samples=nbr échantillons
    n_samples = np.shape(data)[0]
    clusters = [[] for _ in range(self.k)]
    for sample_i, sample in enumerate(data):
        centroid_i = self._closest_centroid(sample, centroids)
        clusters[centroid_i].append(sample_i)
    return clusters
```

Calculer de nouveaux centroïdes comme moyennes des échantillons dans chaque cluster :

```python
# Calculer de nouveaux centroïdes comme moyennes des échantillons dans chaque cluster
def _calculate_centroids(self, clusters, data):
    n_features = np.shape(data)[1]
    centroids = np.zeros((self.k, n_features))
    for i, cluster in enumerate(clusters):
        #Calculer la moyenne arithmétique le long de l'axe 0
        centroid = np.mean(data[cluster], axis=0)
        centroids[i] = centroid
    return centroids
```

Effectuez un clustering K-Means et renvoyez les centroïdes des clusters :

Nous pouvons maintenant répéter les étapes d'attribution de points aux clusters et de mise à jour itérative des centres de cluster et observer la progression de l’algorithme.

Après chaque itération, tester si un point modifie son appartenance au cluster.

Arrêtez l'algorithme si la convergence a été atteinte, c'est-à-dire que les clusters ne changent pas après l'étape de réallocation.

```python
# Effectuez un clustering K-Means et renvoyez les centroïdes des clusters
def fit(self, data):
    # 1-Initialiser les centroïdes
    centroids = self._init_random_centroids(data)

    # Itérer jusqu'à convergence c.à.d pas de changement des centres
    for _ in range(self.max_iterations):
        # 2-Affecter des échantillons aux centroïdes les plus proches c.à.d créer des clusters
        clusters = self._create_clusters(centroids, data)

        prev_centroids = centroids
        # 3-Calculer de nouveaux centroïdes à partir des clusters
        centroids = self._calculate_centroids(clusters, data)

        # Si aucun barycentre n'a changé => Critère d'arret
        diff = centroids - prev_centroids
        if not diff.any():
            break

    self.kmeans_centroids = centroids
    return centroids
```

```python
# Prédire la classe d'un échantillon
def predict(self, data):

    # Vérifiez d'abord si nous avons déterminé les centroïdes K-Means
    if not self.kmeans_centroids.any():
        raise Exception("K-Means centroids have not yet been determined")

    clusters = self._create_clusters(self.kmeans_centroids, data)

    predicted_labels = self._get_cluster_labels(clusters, data)

    return predicted_labels
```

Trouver les centres des 3 clusters :

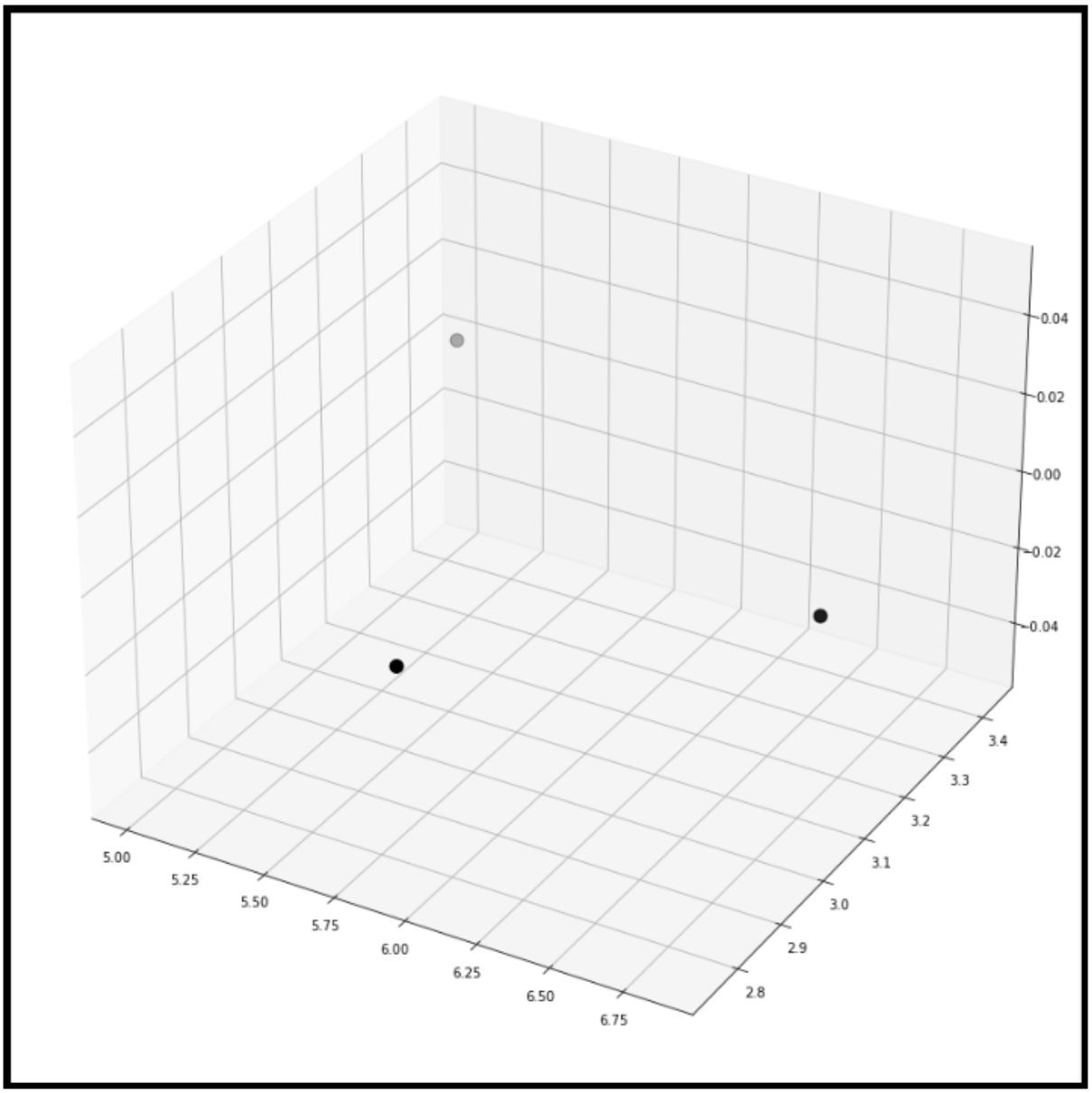
```python
centroids = clf.fit(train_data)
```

```python
centroids
```

```
array([[6.85      , 3.07368421, 5.74210526, 2.07105263],
       [5.006     , 3.428     , 1.462     , 0.246     ],
       [5.9016129 , 2.7483871 , 4.39354839, 1.43387097]])
```

Affichons les centroïdes des clusters :

```python
#Plotting the centroids of the clusters
fig = plt.figure(figsize = (15,15))
ax = fig.add_subplot(111, projection='3d')
plt.scatter(centroids[:, 0], centroids[:,1], s = 100, c = 'black', label = 'Centroids')
plt.show()
```



Classer les points de données dans les 3 clusters :

```
predicted_labels = clf.predict(train_data)
```

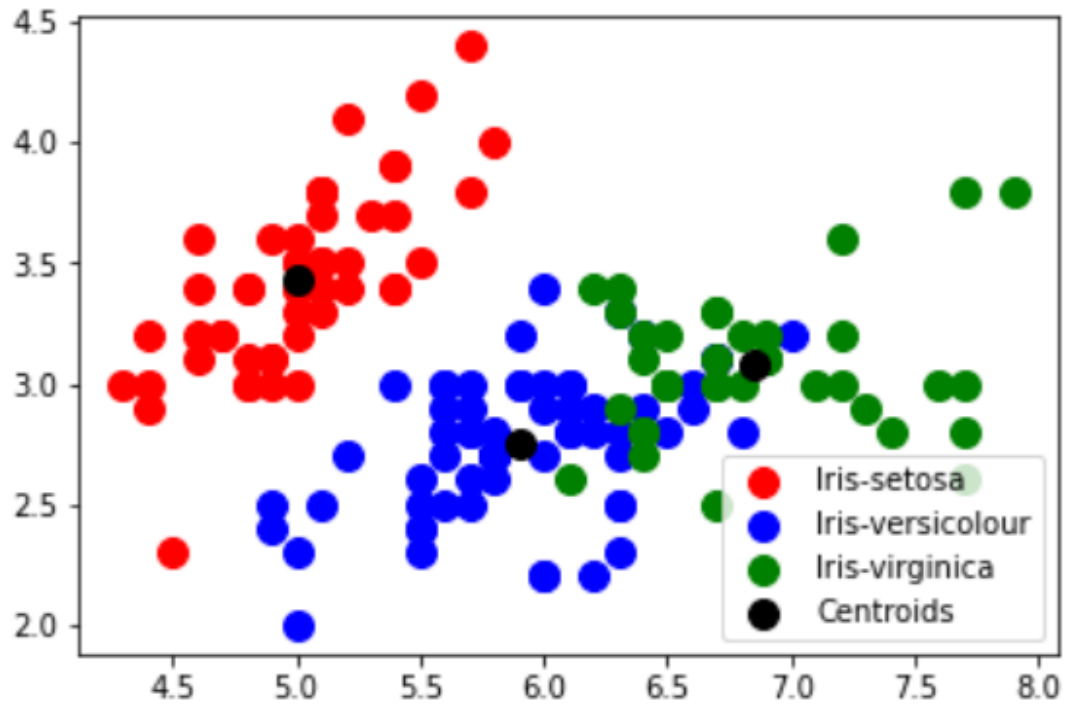
Visualiser la répartition des fleurs par clusters :

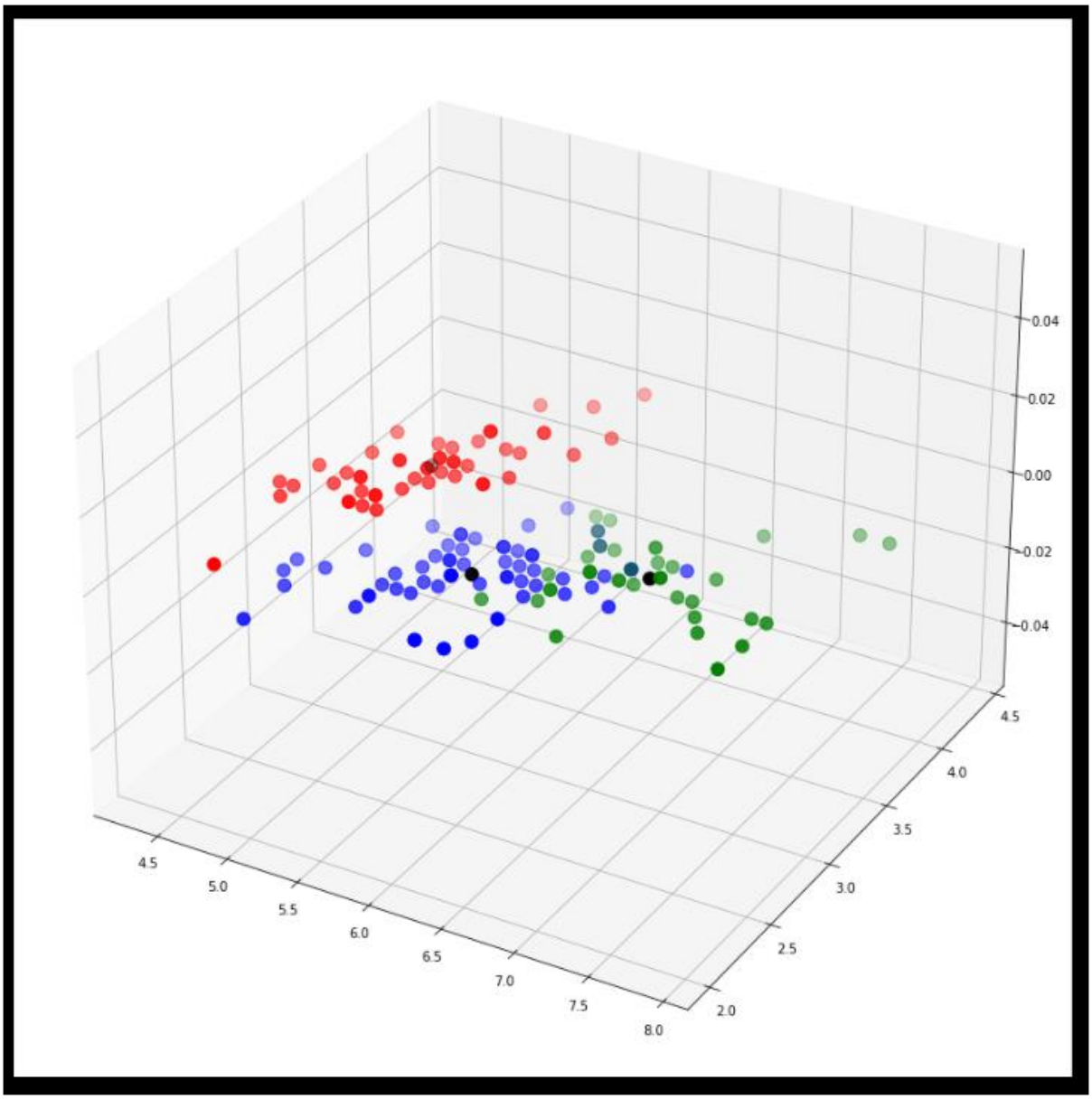
```
# Visualising the clusters - On the first two columns
plt.scatter(x[predicted_labels == 0, 0], x[predicted_labels == 0, 1],
            s = 100, c = 'red', label = 'Iris-setosa')
plt.scatter(x[predicted_labels == 1, 0], x[predicted_labels == 1, 1],
            s = 100, c = 'blue', label = 'Iris-versicolour')
plt.scatter(x[predicted_labels == 2, 0], x[predicted_labels == 2, 1],
            s = 100, c = 'green', label = 'Iris-virginica')

# Plotting the centroids of the clusters
plt.scatter(centroids[:, 0],centroids[:,1],
            s = 100, c = 'black', label = 'Centroids')

plt.legend()
```

```
<matplotlib.legend.Legend at 0x26c8c1fc8b0>
```




0.04
0.02
0.00
-0.02
-0.04
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0