Master 1 SD

Analyse et visualisation de données

Nom : Niang
Prénom : Souna

Séance de TP 2
K-means, K-médoïdes, critères de qualité

1- Programmer la méthode des k-moyennes

Rappeler l'algorithme et décrire les choix d'implémentation en termes de structure et types de données manipulées et passés en argument d'entrées et en valeur de retour de méthode.

Donner votre programme en annexe.

2- Examiner la stabilité de l'algorithme sur plusieurs exécutions

Pour comparer les exécutions pour une même valeur de K, on s'aidera des critères suivants :

- Nombre d'itérations jusqu'à converge pour un seuil de 0,001
- Pourcentage de variance expliquée
- Position des centroïdes

- Effectuer cette analyse pour le dataset IRIS et pour le dataset Breast Cancer

3- Initialisation k-means++

Rappeler le principe de la méthode d'initialisation k-means++

Décrire vos choix d'implémentation en termes de structure et type de données manipulées et passés en arguments d'entrée et en valeurs de retour de méthode

Donner votre programme en annexe.

4- Examiner la stabilité de l'algorithme avec kmeans++

Pour comparer les exécutions pour une même valeur de K, on s'aidera des critères suivants :

Nombre d'itérations jusqu'à converge pour un seuil de 0,001

Pourcentage de variance expliquée

Position des centroïdes

- Effectuer cette analyse pour le dataset IRIS et pour le dataset Breast cancer
- Comparer ces résultats avec ceux obtenus à la question 2

5- Programmer la méthode des k-médoïdes++

Rappeler l'algorithme et décrire les choix d'implémentation en termes de structure et types de données manipulées et passés en argument d'entrées et en valeur de retour de méthode.

Donner votre programme en annexe.

6- Examiner la stabilité de l'algorithme avec k-medoïdes+

Pour comparer les exécutions pour une même valeur de K, on s'aidera des critères suivants

Nombre d'itérations jusqu'à converge pour un seuil de 0,001

Pourcentage de variance expliquée

Position des centroïdes

- Quel est l'apport de l'initialisation k-means++ sur les k-medoïdes ?
- Effectuer cette analyse pour le dataset IRIS et pour le dataset Breast cancer

7- Vérifier le bon fonctionnement de vos programmes en les comparant aux méthodes de scikit learn

Vous choisirez un exemple d'exécution et comparerez les résultats obtenus par votre méthode et la méthode équivalente de scikit-learn.

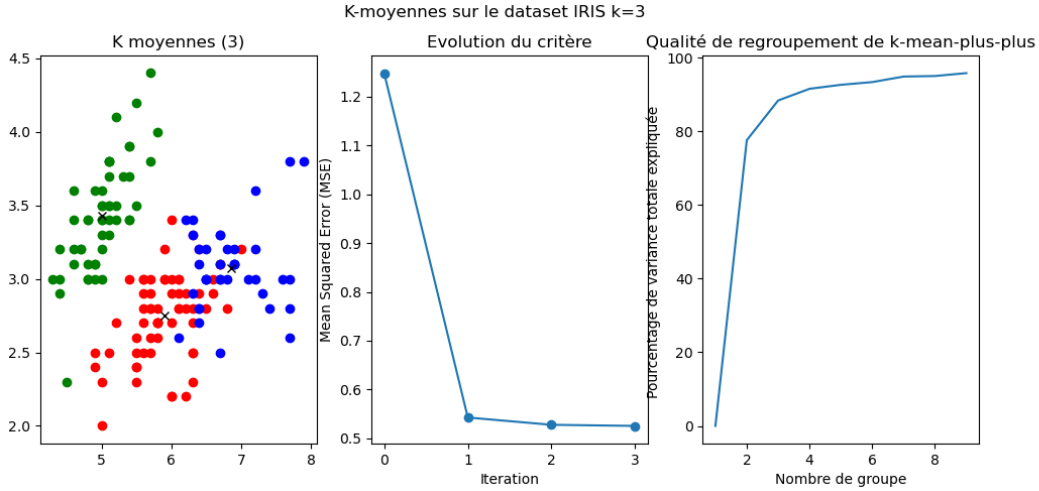
RÉPONSE : ALGORITHME K-MOYENNE

L'algorithme de **k-moyennes** est un algorithme de classification non supervise qui pour un jeu de données X et un entier k (2<=k<=len(X)) classe les éléments de X en K groupe avec des variances intra-groupes faibles et une variance inter-groupes élevée.
Initialement l'algorithme choisit k éléments de X de manière ***aléatoire*** pour représenter les centroïdes des k groupes ensuite il affecte chaque élément de X au groupe ayant le centroide le plus proche de lui en calculant les distances ensuite l'algorithme va calculer la moyenne de chaque groupe et va considérer cette valeur comme nouveau centroide du groupe. L'algorithme répètera ce processus jusqu'à ce que la différence de l'Erreur quadratique moyenne ou Mean Squared Error(MSE) en anglais entre deux itérations successives soit inférieur au seuil ou jusqu'à ce que l'on atteint un nombre maximal d'itération.
La méthode de **k-moyennes** ci-dessous prend en paramètre une matrice **X** qui représente les données, un entier **k** qui représente le nombre de groupe qu'on veut avoir à la fin, un nombre décimal nommé **seuil** qui permet de tester le critère quadratique et un nombre entier nommé **Max_iter** qui représente le nombre d'itération maximal. A la fin de l'exécution la méthode retourne une matrice **C** dont chaque colonne représente un centroide, un vecteur **y** dont la taille est égale à len(X) et que y[i] représente le groupe de X[i] et un vecteur J dont la taille est égale au nombre d'itérations et représente le MSE de chaque itération .

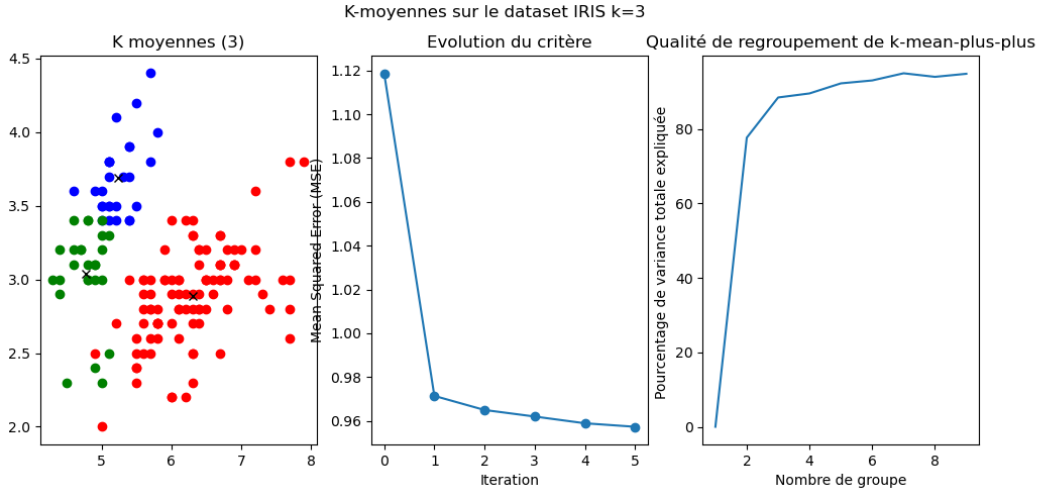
2)Pour tester la stabilité de l'algorithme k-moyennes je vais l'exécuter deux fois avec k=3

-Avec iris

1ere exécution

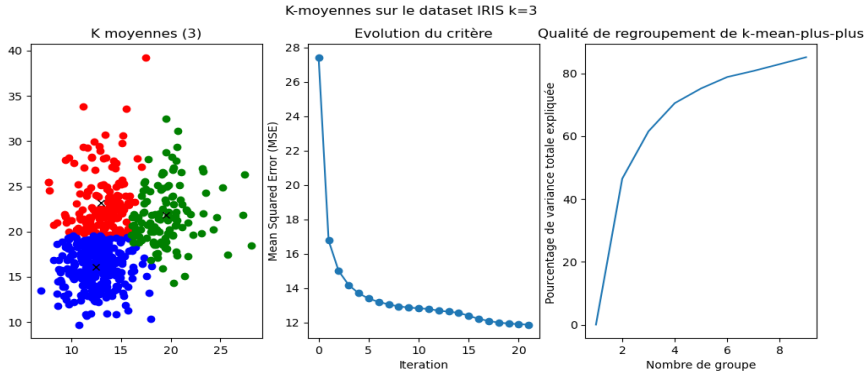


2e exécution

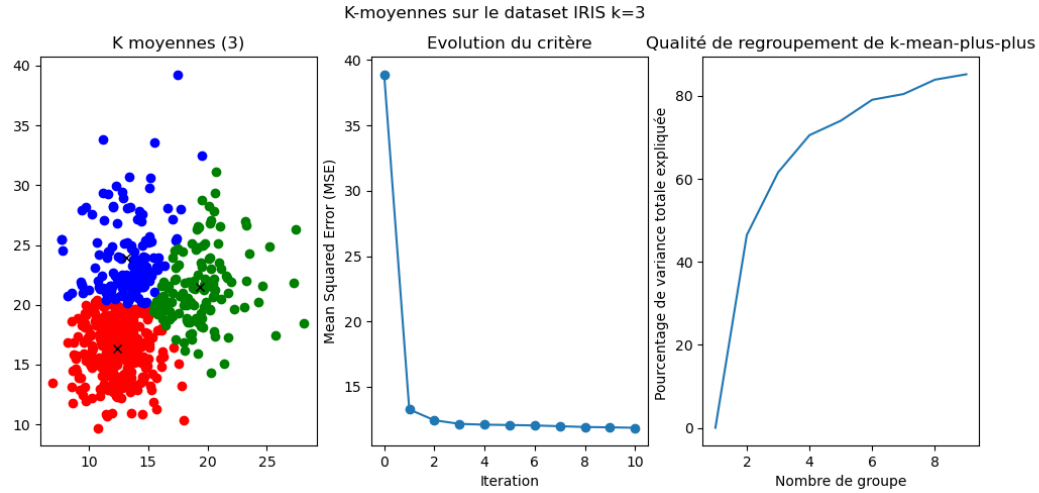


-Brest cancer

1ere exécution



2e exécution



Sur les deux datasets on voit que les positions de centroides changent. Cette instabilité du k- moyennes est due au tirage aléatoire des clusters à l'initialisation.
On voit que le nombre d'itération change d'une exécution à l'autre. La troisième figure permet de visualiser l'évolution du pourcentage de la variance totale expliquée en fonction du nombre de groupe et cela permet de voir le nombre de groupe optimal.

Pour choisir un k optimal j'ai exécuté l'algorithme sur plusieurs valeurs de k(3e figure) et ensuite j'ai calculé le pourcentage de variance totale expliquée pour chaque valeur de k. Avec le dataset iris on voit que pour les valeurs de k<3 le pourcentage est un peu faible (si k=1 le pourcentage est nul et on ne veut pas ça) alors que pour celles >3 le pourcentage est très élevé (si k=len(X) le pourcentage est nul et on ne veut pas ça non plus) donc la valeur optimale est k=3 par contre avce le data set Breast cancer la valeur optimale est ici k=5

RÉPONSE : ALGORITHME K-MOYENNE PLUS-PLUS

L'algorithme K-moyenne Plus-Plus est une variante de l'algorithme K-moyenne la seule différence est que K-moyenne-plus-plus diminue l'effet aléatoire de K-moyenne c'est-à-dire au lieu de choisir des clusters de manière aléatoire lors de la première itération K-moyenne-plus-plus choisit des centroide très éloigne en se basant sur des probabilité éléments. Concernant les choix d'implémentation en terme de structure de données ou de paramètre d'entre et de sortie K-moyenne-plus-plus à la même configuration que K-moyenne sauf l'ajout d'une fonction nommée **initPlus** qui prend en paramètre une matrice X représentant les données et un entier K représentant le nombre de groupe et retourne la fin une matrice **C** qui contient les centroide lors à l'initialisation

-Avec iris

1ere exécution

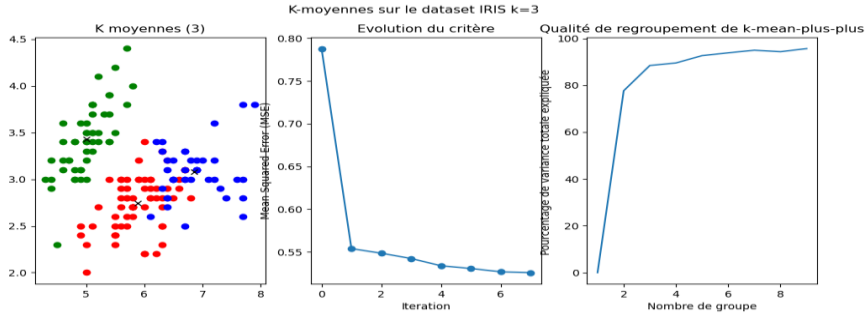
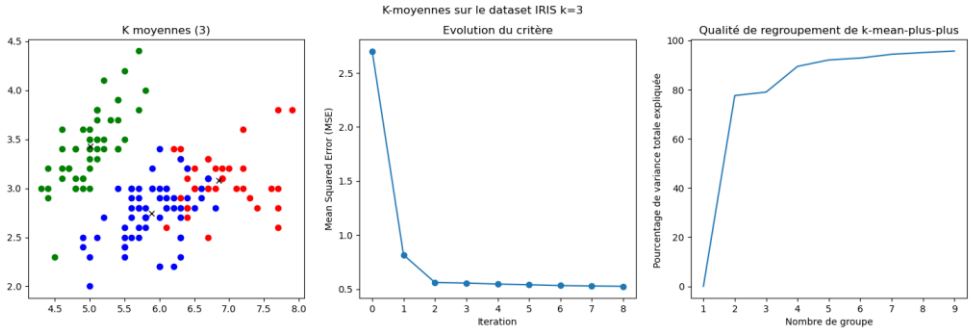


Figure 1 Resultat de classification de k-moyennes-plus-plus sur le dataset iris avec k=3

2e exécution

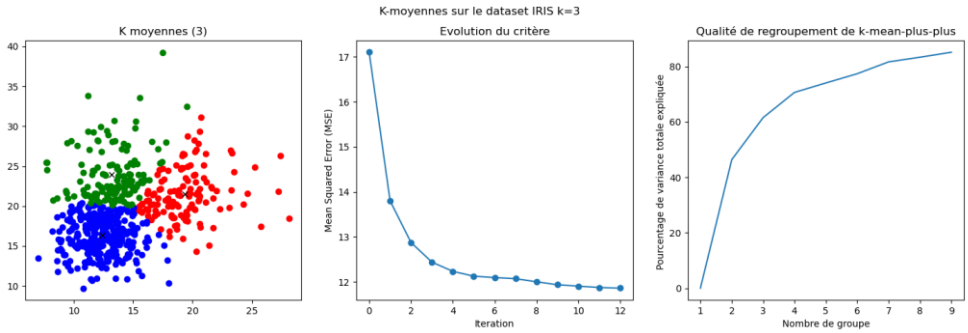


On voit que les positions de centroides changent de manière moins significatis contrairement avec k-moyenne.

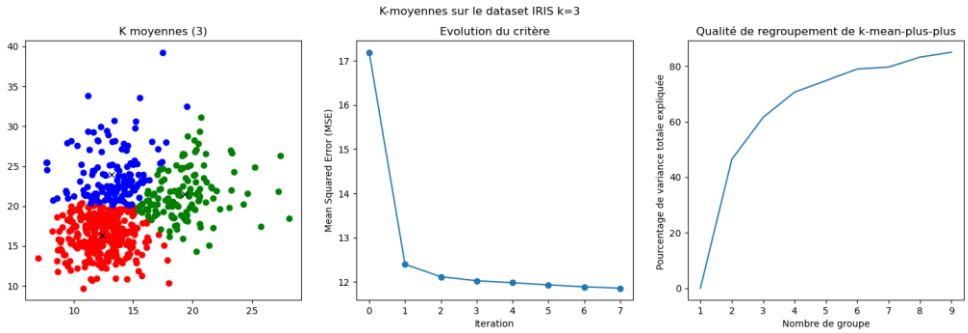
On voit que le nombre d'itération est plus petit dans la deuxième exécution que sur la première.

-Brest cancer

1ere exécution



2e exécution



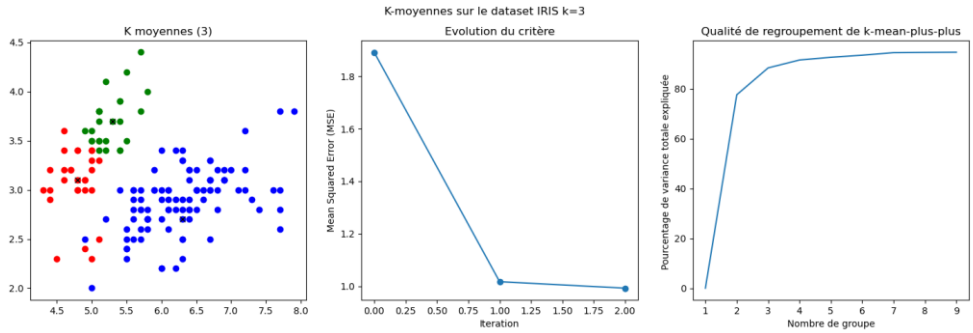
Dans les deux datasets on voit que les positions des centroides ne changent pas ou bien très peu dans ce cas-ci, le nombre d'itération est plus grand dans la deuxième exécution que sur la première. On voit bien l'initialisation de l'algorithme de k-moyennes++ qui permet de contrôler l'aspect aléatoire de k-moyennes diminue l'instabilité
En se basant sur le pourcentage de la variance expliquée, la valeur de K optimal est égale 3 pour iris et 5 pour Breast cancer

RÉPONSE : ALGORITHME K-MÉDOÏDES++

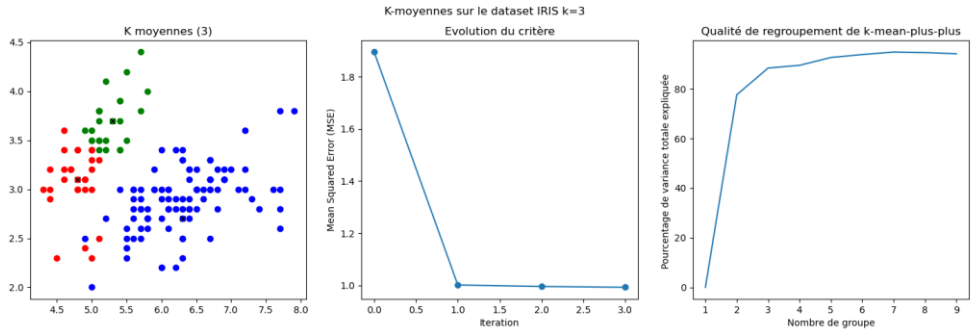
L'algorithme k-médoïdes++ est une variante de l'algorithme K-moyenne++ la seule différence est qu'au lieu de prendre la moyenne de l'ensemble des elements d'un groupe comme centroide, k-médoïdes++ prend l'élément dont la somme de ses distances par rapport aux autres éléments du groupe est la plus faible. Concernant les choix d'implémentation en terme de structure de données ou de paramètre d'entre et de sortie k-médoïdes++ à la même configuration que K-moyenne++ sauf le calcul des distances 2 à 2 de chaque groupe

-Avec iris

1ere exécution

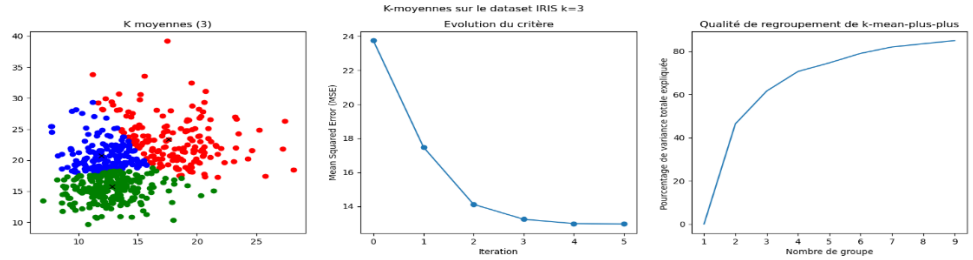


2e exécution

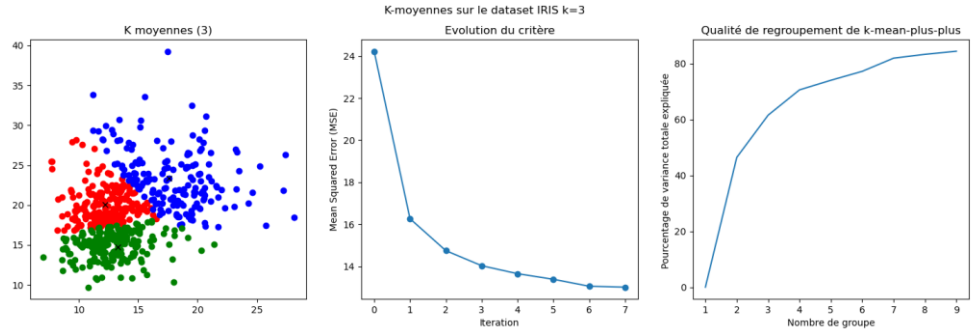


-Brest cancer

1ere exécution



2e exécution



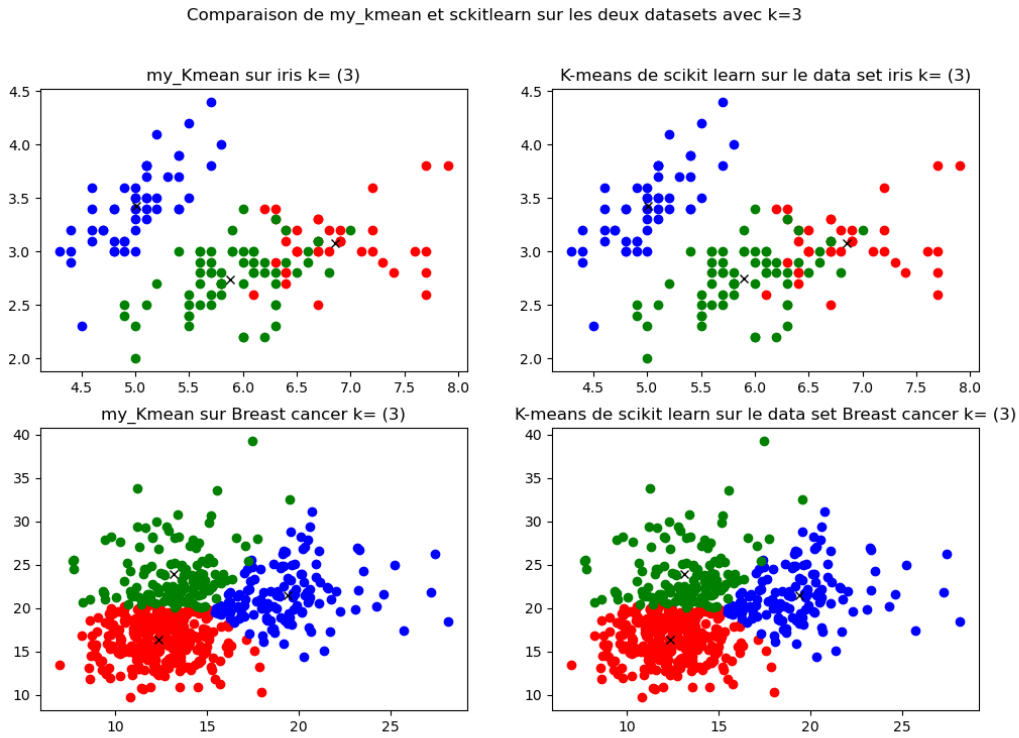
Dans les deux datasets on voit que les positions des centroides ne changent pas ou bien très peu comme k-moyennes++, on peut noter aussi que sur le dataset iris le nombre d'itération est plus faible avec l'algorithme des k-medoide++ par rapport à k-moyennes ou k-moyennes++. On voit bien comme avec k-moyennes++ l'initialisation de l'algorithme permet de contrôler l'aspect aléatoire et diminue l'instabilité.
La diminution du nombre d'itération est due au fait que le k-medoides ne prend pas la moyenne des éléments comme centroides mais plutôt l'élément le plus proches des tous les autres et comme ça tous les centroides se trouve dans X

En se basant sur le pourcentage de la variance expliquée, la valeur de K optimal est égale 3 pour iris et 5 pour Breast cancer

COMPARAISON AVEC LES METHODES DE SKCITLEARN

- **K-MEAN**

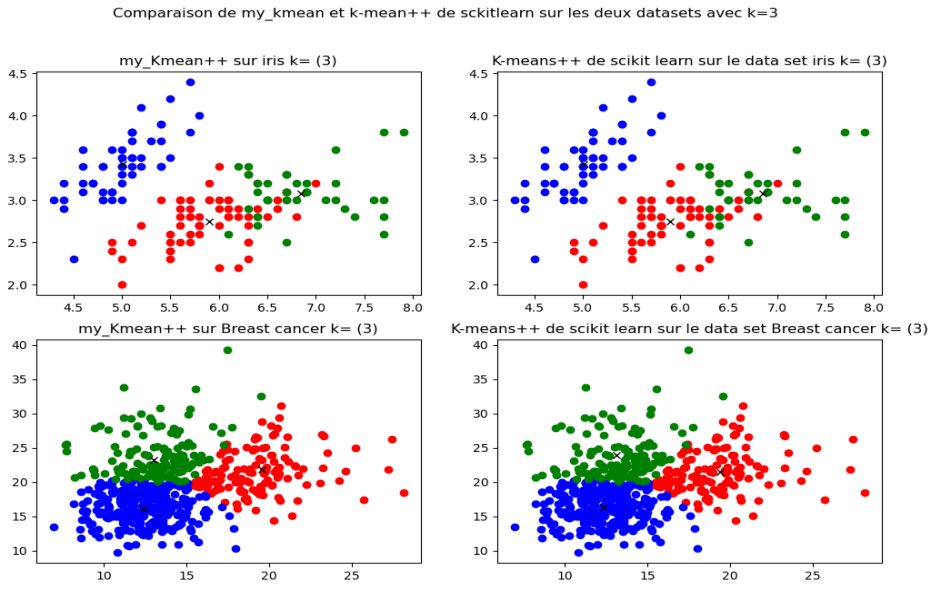
  Comparaison entre l'algorithme K-means et celui de sckitlearn



On les même centroide donc ma méthode fonctionne

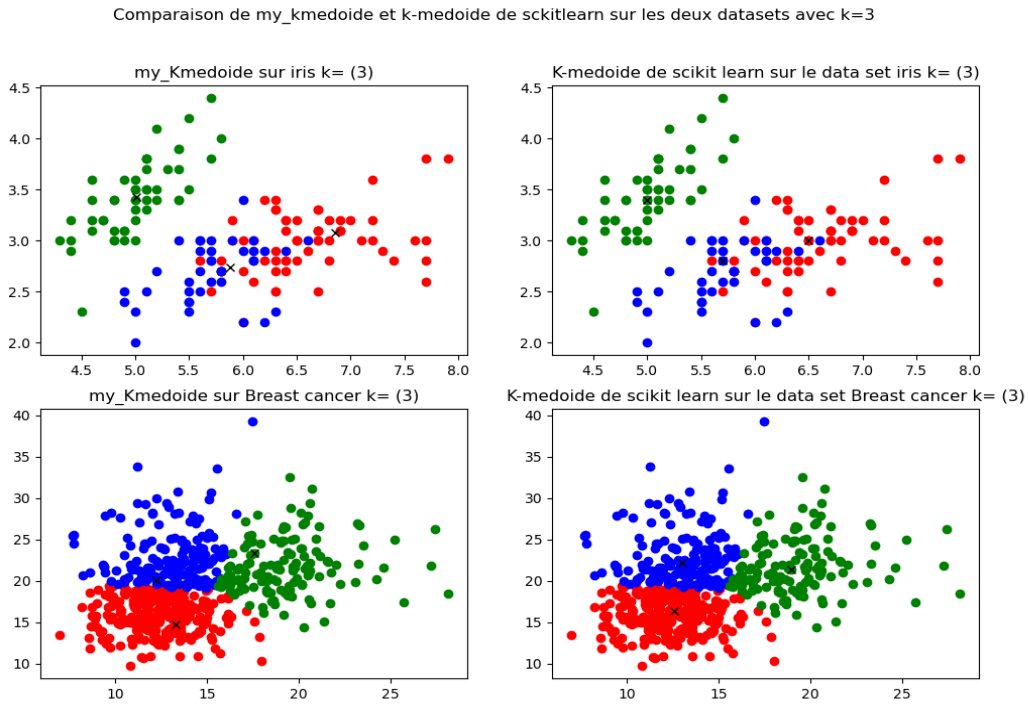
- **K-MEAN++**

  Comparaison entre l'algorithme K-means++ et celui de sckitlearn



Ils ont les même centroide

- **K-MEDOIDES**
  Comparaison entre l'algorithme K-medoides et celui de sckitlearn



Les centroides sont un peu differents

Annexe

K-Means

```
def my_kmeans(X,K,Visualisation=False,Seuil=0.001,Max_iterations = 100000):

    N,p = np.shape(X)
    iteration = 0
    Dist=np.zeros((K,N))
    J=np.zeros(Max_iterations+1)
    J[0] = 10000000

    # Initialisation des clusters
    # par tirage de K exemples, pour tomber dans les données

    Index_init = np.random.choice(N, K,replace = False)
    C = np.zeros((p,K))
```

```
    for k in range(K):
        C[:,k] = X[Index_init[k],:].T


    while iteration < Max_iterations:
        iteration +=1
        ##########################################################################
        # E step : estimation des données manquantes
        #          affectation des données aux clusters les plus proches
        for k in range(K):
            Dist[k,:]=np.linalg.norm(X-C[:,k],axis=1)**2
        y=np.argmin(Dist,axis=0)
        ##########################################################################
        # M Step : calcul des meilleurs centres
        for k in range(K):
             C[:,k]=np.mean(X[y==k,:],axis=0)
        # test du critère d'arrêt l'évolution du critère est inférieure
        # au Seuil en pour ceent
        J[iteration]=np.sum(np.min(Dist[y,:],axis=0))/N

        if np.abs(J[iteration]-J[iteration-1])/J[iteration-1]<Seuil:
            break;
    return C, y,J[1:iteration]
```

K-mean++

```
def initPlusPlus(X,K):
    N,p = np.shape(X)
    C = np.zeros((p,K))
    generator = np.random.default_rng()
    index = np.random.choice(N, 1,replace = False)
    liste_index = [index]
    C[:,0] = X[index,:]
```

```
    X = np.delete(X,index,0)
    # print("k=0 C[k]=",C[:,0],"index=",index)
    k=1

    while k < K:

        # calcul des distances
        NN = X.shape[0]
        dist = np.zeros(NN)
        for n in range(NN):
            D=C[:,:k]-np.repeat(X[n,:],k).reshape(p,k)
            D=np.diag(D@D.T)
            dist[n]=np.min(D)
       # ICI .....
        # calcul des probabilités
        proba=dist/np.sum(dist)
        range_value=generator.random((1))[0]
        intervals=np.cumsum(proba)
        index=0
        while index<NN:
            if intervals[index]>range_value:
                break;
            index+=1
        # ICI ....

        # tirage aléatoire selon proba
        C[:,k]=X[index,:]
        X=np.delete(X,index,0)
        k+=1
       # ICI .....
    return C



#----------------K-mean++-------------------------------

def my_kmeans_plus_plus(X,K,Visualisation=False,Seuil=0.001,Max_iterations = 1000):
```

```
    N,p = np.shape(X)
    iteration = 0
    Dist=np.zeros((K,N))
    J=np.zeros(Max_iterations+1)
    J[0] = 10000000
    C=initPlusPlus(X,K)

    while iteration < Max_iterations:
        iteration +=1

        # E step : estimation des données manquantes
        #          affectation des données aux clusters les plus proches
        for k in range(K):
            Dist[k,:]=np.linalg.norm(X-C[:,k],axis=1)**2
        y=np.argmin(Dist,axis=0)
        # M Step : calcul des meilleurs centres
        for k in range(K):
             C[:,k]=np.mean(X[y==k,:],axis=0)
        ###########################################################################
        # test du critère d'arrêt l'évolution du critère est inférieure
        # au Seuil en pour ceent
        J[iteration]=np.sum(np.min(Dist[y,:],axis=0))/N

        if np.abs(J[iteration]-J[iteration-1])/J[iteration-1]<Seuil:
            break;
    return C, y,J[1:iteration]
```

K-medoide

```
def my_kmean_medoide(X,K,Visualisation=False,Seuil=0.0001,Max_iterations = 1000):

    N,p = np.shape(X)
    iteration = 0
    Dist=np.zeros((K,N))
```

```
    J=np.zeros(Max_iterations+1)
    J[0] = 10000000

    # Initialisation des clusters
    # par tirage de K exemples, pour tomber dans les données

    C=initPlusPlus(X,K)

    while iteration < Max_iterations:
        iteration +=1
        # E step : estimation des données manquantes
        #          affectation des données aux clusters les plus proches
        for k in range(K):
            Dist[k,:]=np.linalg.norm(X-C[:,k],axis=1)**2
        y=np.argmin(Dist,axis=0)
        # M Step : calcul des meilleurs centres
        for k in range(K):
            c_k=X[y==k]
            dist=np.array([sum([np.linalg.norm(c_k[j]-c_k[i])**2 for i in range(len(c_k))]) for j in
range(len(c_k))])
            C[:,k]=c_k[np.argmin(dist)]
        ##########################################################################
        # test du critère d'arrêt l'évolution du critère est inférieure
        # au Seuil en pour ceent
        J[iteration]=np.sum(np.min(Dist[y,:],axis=0))/N

        if np.abs(J[iteration]-J[iteration-1])/J[iteration-1]<Seuil:
            break;
    return C, y,J[1:iteration]
```

La methode qui calcule le critere de regroupement d'une methode en fonction du nombre de groupe

```
def qualite_regroupement(X,methode,k):
```

```
 Cluster, y, Critere = methode(X,k,Visualisation = False)
Ik=np.array([np.sum((X[y==k]-Cluster[:,k])**2)/len(X[y==k]) for k in range(len(Cluster[0]))])
Iw=np.sum([(len(X[y==k])*Ik[k])/len(X) for k in range(k)])
Ib=np.sum([(len(X[y==k])/len(X))*(X.mean(axis=0)-Cluster[:,k])**2 for k in range(k)])
It=Iw+Ib
C=100*(1-(Iw/It))
return C,k
```