Veysseire Daniel TP1 IAA Université Paul Sabatier

TP1

IAA

Tests

2.1) La réallocation :

Pour cette fonction, on peut essayer de faire des exos corrigés du cours et voir si on trouve les mêmes résultats.

Soit l'ensemble de points suivants de R^2 :

Y={(1,1),(3,3),(1,5),(-3,5),(-5,3),(-3,1),(3,-1),(5,-3),(3,-5),(-1,-5),(-3,-3),(-1,-1)}

On avait pris x1=(1,1) , x2=(3,3)....

avec comme dictionnaire

$$D_0^0 = \{(1,2), (-2,-1)\}$$

et on avait trouvé deux partitions :

R1={x5,x6,x9,x10,x11,x12}

R2={x1,x2,x3,x4,x7,x8}

Cela tombe bien, je viens de remarquer que c'est justement cet exemple qui a été utilisé comme test sur cette fonction.

Le résultat trouvé est :

```
reallocation: [[[1.0, 1.0], [3.0, 3.0], [1.0, 5.0], [-3.0, 5.0], [3.0, -1.0], [5
.0, -3.0]], [[-5.0, 3.0], [-3.0, 1.0], [3.0, -5.0], [-1.0, -5.0], [-3.0, -3.0],
[-1.0, -1.0]]]
```

ce qui correspond bien à ce qu'on a trouvé en cours.

<u>Test des cas extrêmes :</u>

Si le nombre de classes est supérieur au nombre de points, ou si le nombre de classe désiré est inférieur ou égale à 0, on a une erreur de type

AssertionError:

grâce aux lignes

```
# on verifie que le nombre de classes désiré est inférieur au nombre de point
        assert(nbGroupes <= len(data))
#on verifie que le nombre de classes désiré est supérieur à 0
        assert(nbGroupes > 0)
```

2.2) Actualisation des centroïdes

Pour l'actualisation des centroïdes, on peut aussi vérifier avec les résultats calculé en cours sur les même données. Les nouveaux centres doivent être (-5/3,-5/3) et (5/3,5/3)

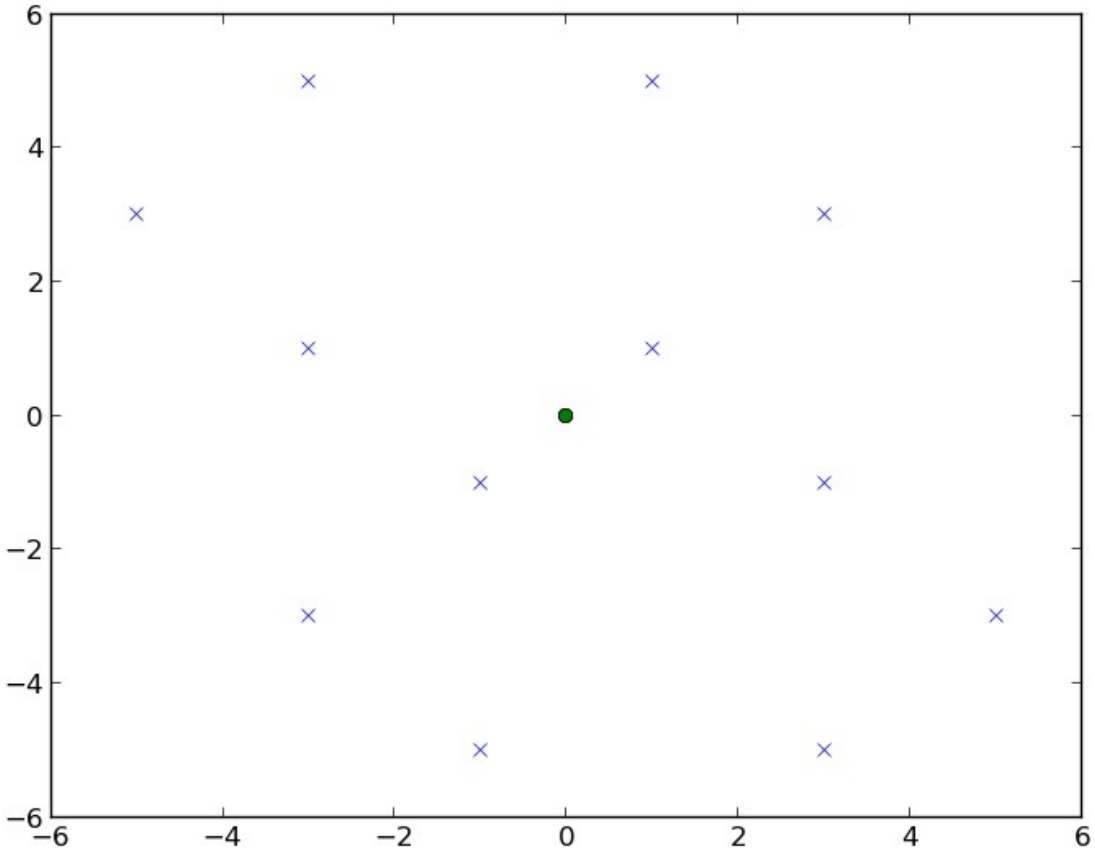
Un print placé après le recentrage renvoie :

```
Centres :
[[1.6666666666666667, 1.6666666666666667], [-1.6666666666666667, -1.666666666666
6667]]
```

Ce qui correspond bien à ce qu'on cherchait.

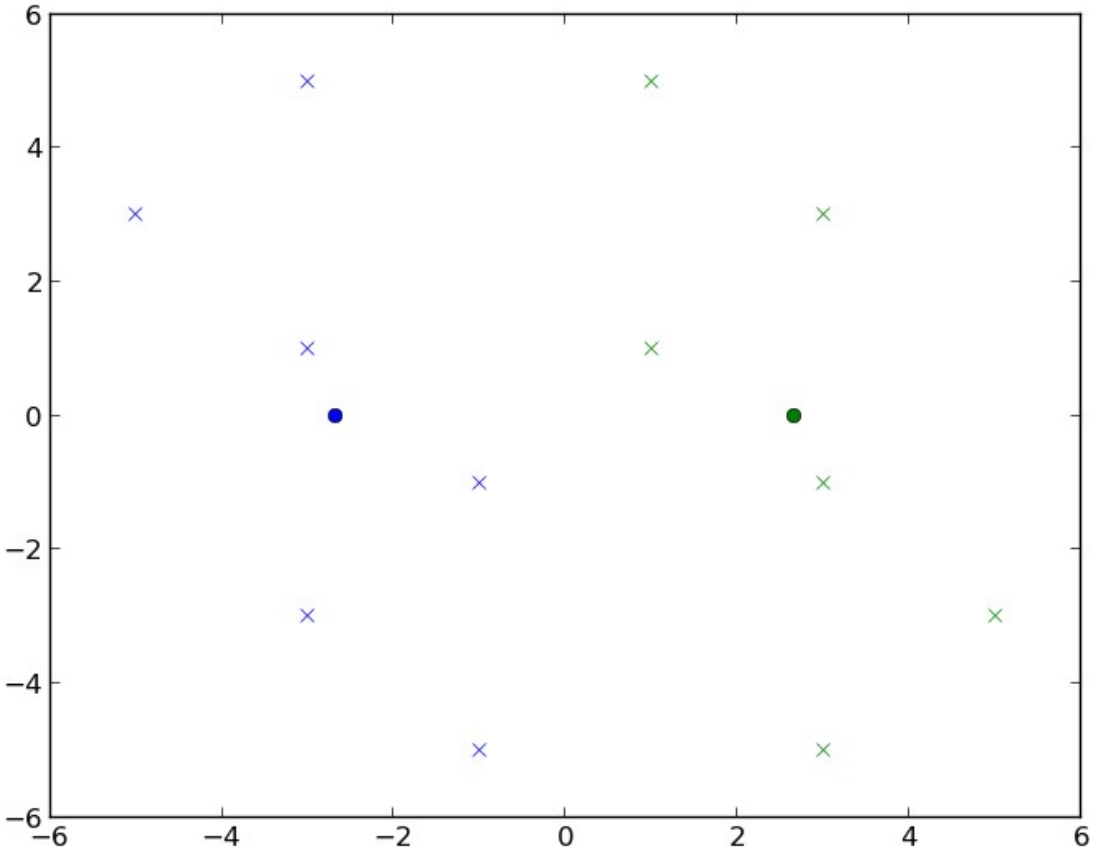
3. L'initialisation aléatoire a bien un effet sur les nuages obtenus. Voici quelques exemples de différences trouvés :

avec D0=[[-3,5], [-3,5]] :



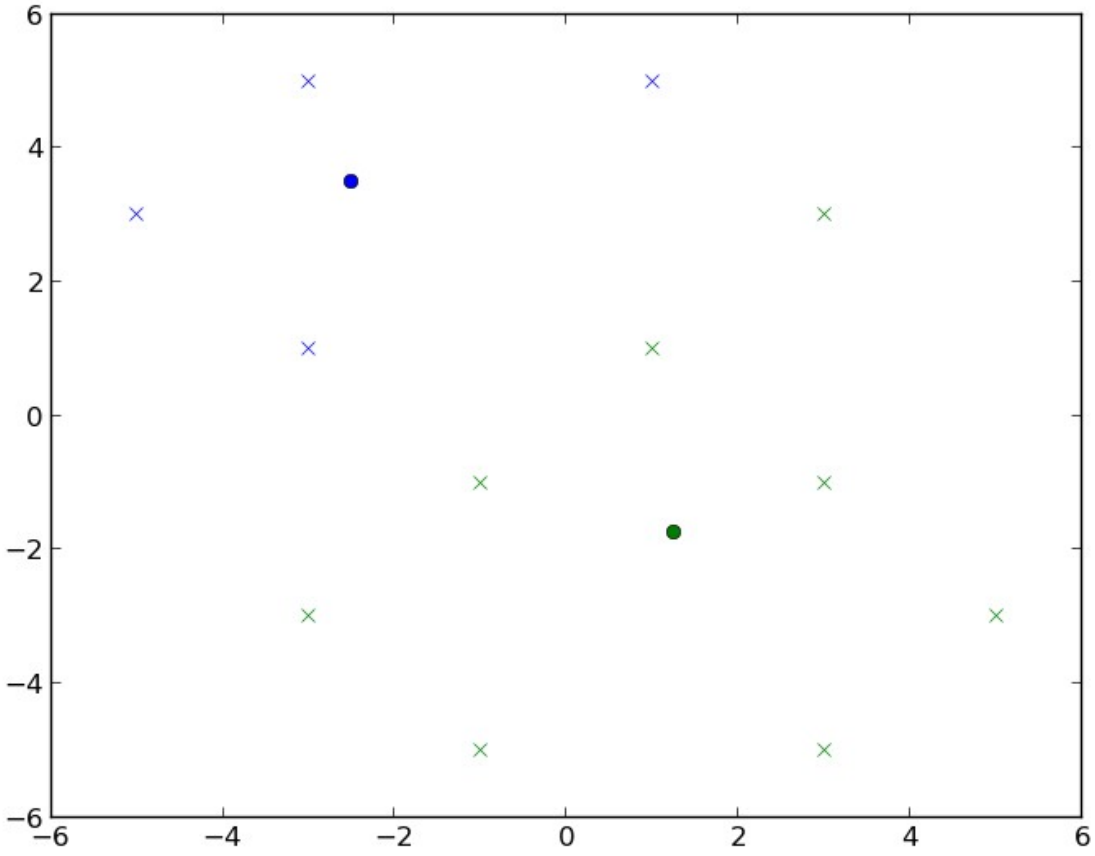
Veysseire Daniel TP1 IAA Université Paul Sabatier

avec [[-5.0, 3.0], [3.0, 3.0]] :



avec [[-3.0, 5.0], [3.0, -1.0]] :

Veysseire Daniel TP1 IAA Université Paul Sabatier



L'initialisation des points a donc bien un impact sur le résultat obtenu.

L'initialisation aléatoire des points n'est pas toujours fiable car on peut avoir deux points confondus comme centre de gravité.

Pour améliorer l'initialisation des centroïdes, on peut utiliser ce qui a été fait au TD1 Q1.2 à savoir séparer le nuages de points en K classes aléatoires contenant le même nombre de points et prendre leurs centres de gravité comme point de départ. Avec cette méthode il est néanmoins très improbable mais possible de se retrouver dans un cas avec les centres de gravités confondus. (exemple 4 points (0,1) (0,-1) (-1,0)(1,0). le centre de gravité de [(0,1) (0,-1)] et celui de [(-1,0)(1,0)] sont 0.

J'ai donc préférer carrément implémenter la classification hiérarchique ascendante par méthode des distances pour avoir K classes, puis prendre leurs centres de gravité comme centres initiaux.

Mon implémentation impose de recalculer une matrice des distances à chaque fusion de régions, ce qui s'avère un peu lourd en calcul mais néanmoins raisonnable pour des petits volumes de données.

Veysseire Daniel TP1 IAA Université Paul Sabatier

Tests

Là aussi on peut comparer les résultats avec ceux obtenus en cours pour l'exo II.1 du tp 1

Étant donné que dans cet exercice on se place en dimension 1 je vais donc mettre des points avec une ordonnée nulle.

obs=[[1,0],[7,0],[15,0],[5,0],[2,0]]

on obtient

```
[[6.0, 14.0, 4.0, 1.0], [8.0, 2.0, 5.0], [10.0, 13.0], [3.0]]
```

Ce qui correspond exactement à la matrice de distance obtenue en TD

Comme nous n'avons pas recalculé la matrice de distances chaque fois en TD mais que je l'ai fait dans le programme, je ne peux pas comparer ces résultats. Néanmoins je peux suivre l'évolution de la fusion des régions et voir si elles correspondent.

```
[[[1, 0], [2, 0]], [[7, 0]], [[15, 0]], [[5, 0]]]
```

puis

```
[[[1, 0], [2, 0]], [[7, 0], [5, 0]], [[15, 0]]]
```

puis

```
[[[1, 0], [2, 0], [7, 0], [5, 0]], [[15, 0]]]
```

Ce qui corrobore bien le résultat trouvé en TD.

On observe que le point (15,0) est isolé

En testant maintenant sur nos données initiales, on obtient aussi une classe avec un point isolé et une classe avec tous les autres points.

[[[1.0, 1.0], [3.0, 3.0], [1.0, 5.0], [3.0, -1.0], [5.0, -3.0], [3.0, -5.0], [-1.0, -1.0], [-3.0, 1.0], [-5.0, 3.0], [-3.0, 5.0], [-3.0, -3.0]] , [[-1.0, -5.0]]]

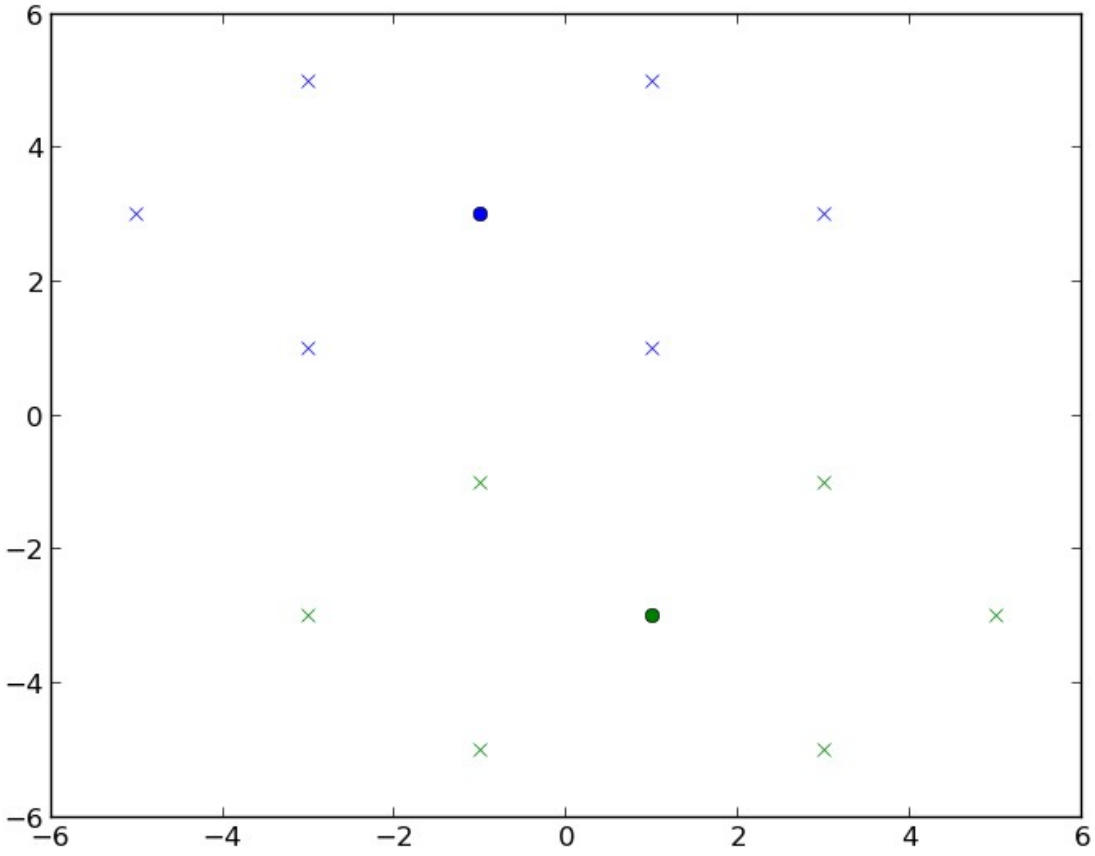
Ceci peut s'expliquer du fait que les points sont globalement à même distance les uns des autres.

La classification hiérarchique ascendante par méthode des distances est donc avantageuses car elle permet d'être sur de ne pas avoir deux fois le même points comme centres, mais elle a un immense inconvénient : il faut que les classes soient assez regroupés et bien séparées pour fonctionner, il suffit d'une donnée erronée extrême pour qu'on se retrouve avec un point isolé comme classe et tous les autres dans l'autre.

Néanmoins avec cette méthode on trouve comme centres

[[0.09090909090909091, 0.45454545454545453], [-1.0, -5.0]]

Ce qui permet, au bout de 5 itérations, de trouver le résultat escompté :



4. Pour compléter et vérifier la notion d'inertie, j'ai aussi implémenté l'inertie totale et l'inertie inter classe en plus de l'inertie intra classe afin de vérifier la décomposition de Huygens.

Tests

En ne mettant aucun coeff à l'inertie intra classe et en faisant comme demandé en commentaire à la fin du fichier kmean, on obtient en sortie :

```
In [12]: run ./kmeansetu.py
calc inertie totale: 20.6666666667
Itération = 1
Données groupées :
[[[1.0, 1.0], [3.0, 3.0], [1.0, 5.0], [-3.0, 5.0], [-5.0, 3.0], [-3.0, 1.0], [3.
0, -1.0], [5.0, -3.0], [3.0, -5.0], [-1.0, -5.0], [-3.0, -3.0], [-1.0, -1.0]], [
]]
Centres :
[[3.0, -5.0], [1.0, 1.0]]
inertie: 656.0
inertie inter: 34.0
veuillez appuyer sur une touche...

old inertie: 656.0 new inertie: 151.111111111 diff_relat: 76.9647696477
Itération = 2
Données groupées :
[[[5.0, -3.0], [3.0, -5.0], [-1.0, -5.0]], [[1.0, 1.0], [3.0, 3.0], [1.0, 5.0],
[-3.0, 5.0], [-5.0, 3.0], [-3.0, 1.0], [3.0, -1.0], [-3.0, -3.0], [-1.0, -1.0]]]
Centres :
[[2.3333333333333335, -4.333333333333333], [-0.7777777777777778, 1.4444444444444
444]]
inertie: 151.111111111
inertie inter: 8.07407407407
veuillez appuyer sur une touche...

old inertie: 151.111111111 new inertie: 137.0 diff_relat: 9.33823529412
```

On obtient bien les résultats attendues. Cependant la décomposition de Huygens n'est pas vérifié ni à l'itération 1 ni à l'it ération 2. En rajoutant cette fois ci le coeff 1/nbPoints à l'inertie intra classe, on obtient cette fois ci :

```
In [13]: run ./kmeansetu.py
calc inertie totale: 20.6666666667
Itération = 1
Données groupées :
[[[1.0, 1.0], [3.0, 3.0], [1.0, 5.0], [-3.0, 5.0], [-5.0, 3.0], [-3.0, 1.0], [3.0, -1.0], [5.0, -3.0], [3.0,
 -5.0], [-1.0, -5.0], [-3.0, -3.0], [-1.0, -1.0]], []]
Centres :
[[3.0, -5.0], [1.0, 1.0]]
inertie: 54.6666666667
inertie inter: 34.0
veuillez appuyer sur une touche...

old inertie: 54.6666666667 new inertie: 12.5925925926 diff_relat: 76.9647696477
Itération = 2
Données groupées :
[[[5.0, -3.0], [3.0, -5.0], [-1.0, -5.0]], [[1.0, 1.0], [3.0, 3.0], [1.0, 5.0], [-3.0, 5.0], [-5.0, 3.0], [-
3.0, 1.0], [3.0, -1.0], [-3.0, -3.0], [-1.0, -1.0]]]
Centres :
[[2.3333333333333335, -4.333333333333333], [-0.7777777777777778, 1.4444444444444444]]
inertie: 12.5925925926
inertie inter: 8.07407407407
veuillez appuyer sur une touche...

old inertie: 12.5925925926 new inertie: 11.4166666667 diff_relat: 9.33823529412
```

A la première itération le théorème n'est toujours pas vérifié. Cependant à la deuxième itération :

inrtie totale = 20.6666666667

inertie intra= 12.5925925926

inertie inter= 8.07407407407

inertie totale= inertie inter+inertie intra

Pourquoi le théorème n'est pas vérifié dès la première itération ? C'est très simple à expliquer, c'est parce que les centres initiaux ne sont pas les centres d'inerties des classes. Résultat, le calcul de l'inertie est faux à la première itération mais vrai à la deuxième.

Il fallait donc bien mettre un coeff 1/nbPoints pour l'inertie intra classe.

Rq : pour l'inertie totale le coeff est lui aussi de 1/nbPoints. Ça se corse avec l'inertie inter classe.

Dans ce cas lors de la somme, chaque région est affecté du coefficient nbPointsRégions/nbPoints.

Remarque : en fait en multipliant tout par nbPoints on obtient la même chose

5. Il y a beaucoup de travaux de recherche , de thèses, sur la recherche du nombre de classes

optimal, la littérature scientifique est très riche à ce sujet.

On ne peut pas utiliser l'inertie totale pour calculer la classe optimal, car cette inertie ne dépends que des points initiaux et non des centres. On peut utiliser comme critère l'inertie intra classe obtenue à la fin. Si on utilise le minimum de l'inertie intra classe on obtient 0 lorsque le nombre de groupes est égale au nombre de classes, et si on utilise le maximum, rebelote on a le cas ou on a une seule classe pour tous les points (en effet l'inertie inter est alors nulle donc inertie intra max)

d'après le poly trouvé sur

http://apiacoa.org/publications/teaching/data-mining/clustering.pdf

p.73 à 77

une bonne classification consiste en :

- classes homogènes

– classes bien séparées

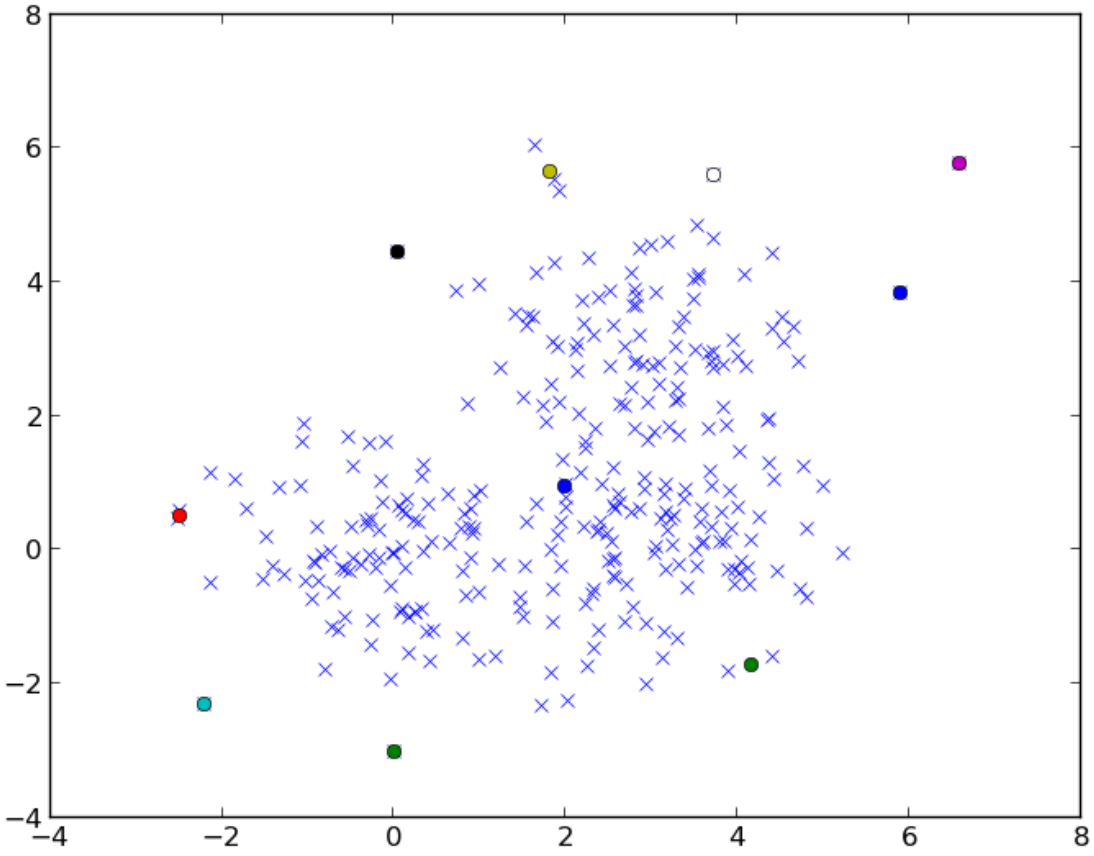
Par manque de temps je n'ai pas implémenté la méthode, mais ce que l'on peut faire est calculer l'inertie intra classe pour chaque K allant de 0 à nbPoints, de tracer la courbe et de prendre le K pour lequel on a un « coude » ou un « pic ».

Remarques :

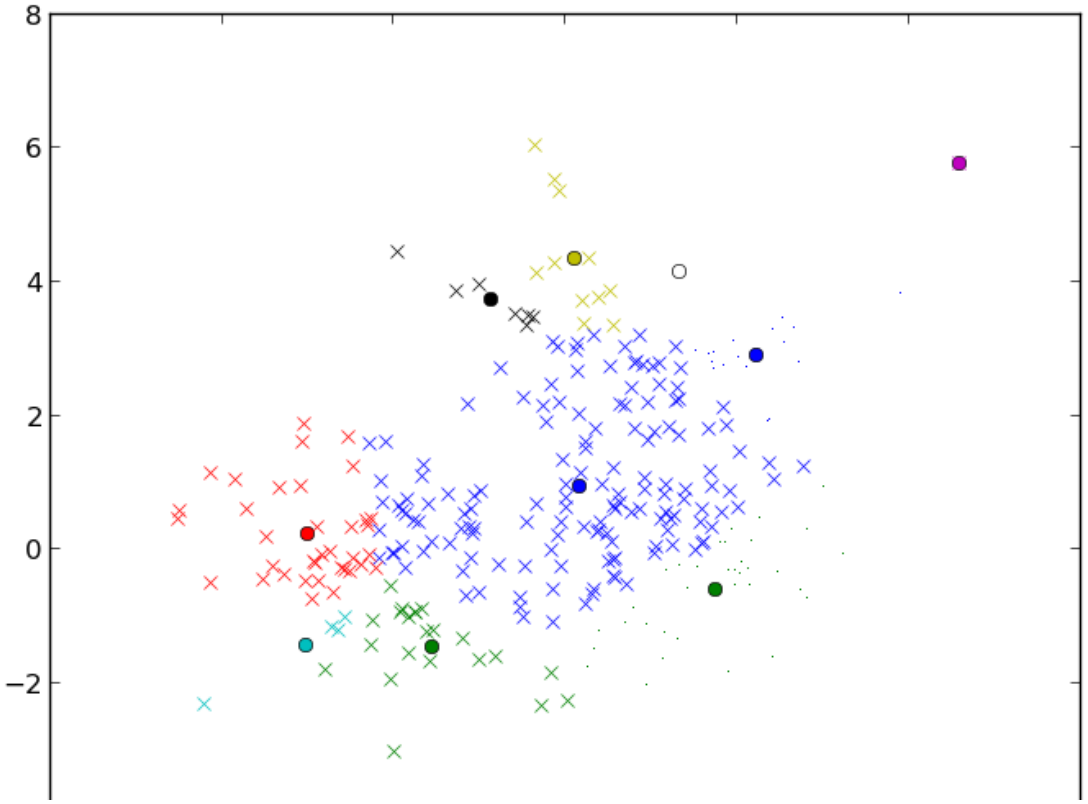
On ne peut pas faire plus de 160 classes car il y a 8 formes et 20 styles. Cependant il aurait été plus judicieux de supprimer 'o' de la liste des styles, car les styles des centres est forcément 'o' se qui fait qu'un point et son centre sont représentés de la même manière ce qui n'est pas une bonne idée. Il est aussi difficile de voir les points blancs sur un fond blanc, on aurait peut êtr du enlever la couleur blanche aussi.

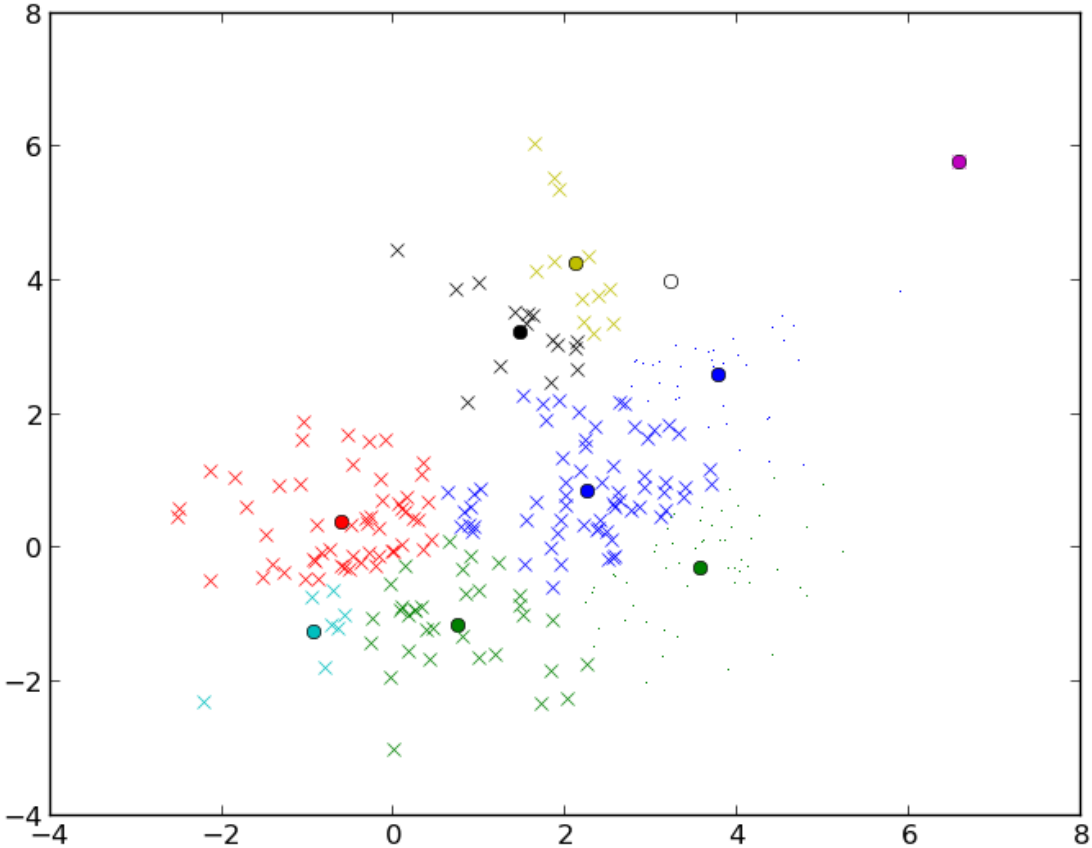
Essaie avec 10 classes pour les données data de data.txt:

Veysseire Daniel TP1 IAA Université Paul Sabatier

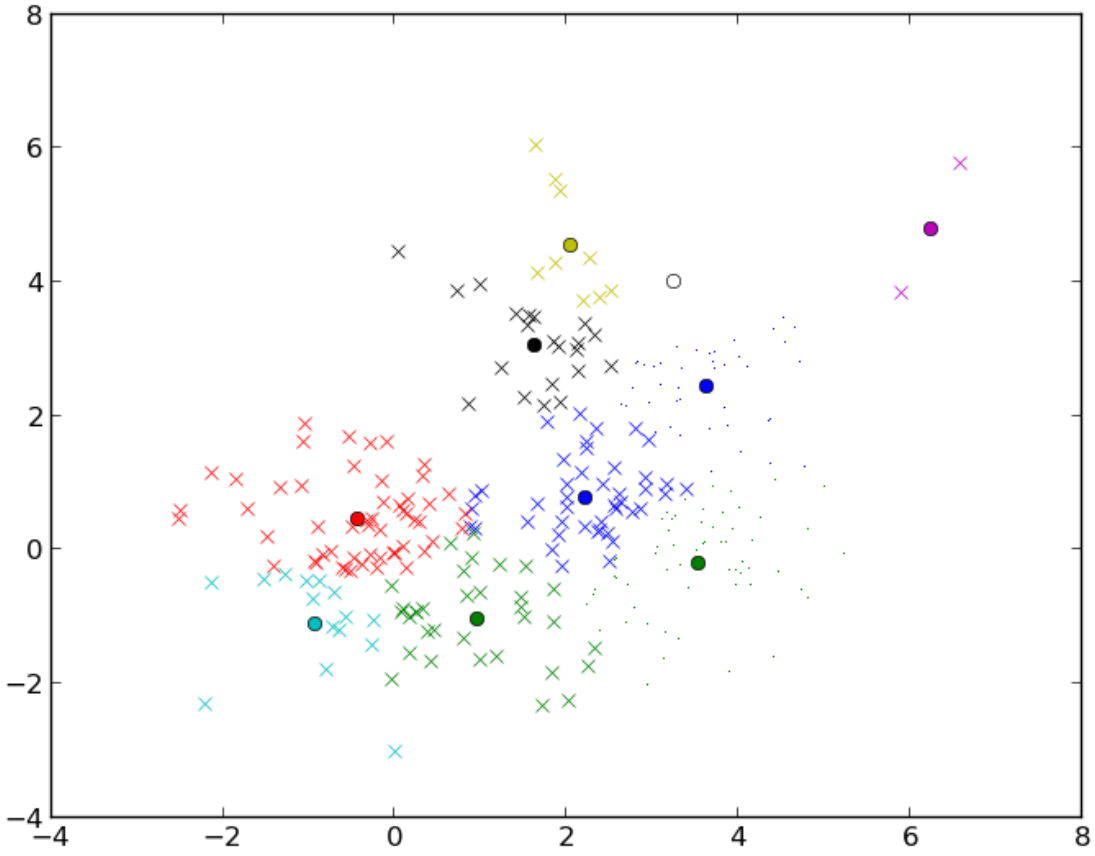


Veysseire Daniel TP1 IAA Université Paul Sabatier





Veysseire Daniel TP1 IAA Université Paul Sabatier



erreurs :

ligne 26:
il faut remplacer

```
if no_couleur > len(couleurs):
```

par

```
if (no_couleur >= (len(couleurs))):
```

car la taille de couleurs est de 8,
donc si on décide de mettre un nombre de groupe égal à 8, on ne passe pas dans la condition et le programme essaie d'accéder à

couleurs[8]
à la ligne 29

rajouts :

`calcul_inertie_totale_nuage(nuage)`
pour calculer l'inertie totale du nuage

`argsMinMatrix(liste)`
pour renvoyer les indices de ligne et de colonne du minimum de la matrice

`calcul_inertie_classe(liste, centre)`
pour calculer l'inertie d'une classe

`calcul_inertie_inter(liste,centre)`
pour calculer l'inertie inter-classe

Incohérence

J'ai noté dans le cours ou TD que l'on pouvait calculer l'inertie totale pour choisir le bon nombre de classe. Or dans la définition que j'ai, l'inertie totale ne dépend pas du nombre de classes et est définie comme étant la somme des carrées des distances de chaque point au centre de gravité de tous les points, le tout affecté d'un coefficient. Cette valeur ne dépend donc que des points et pas du nombre de classes.

pour aller plus loin

Il pourrait être intéressant de coder une fonction qui trace les médiatrices des segments des centres pour voir les demi/plans auxquels appartiennent les points, et voir l'évolution du déplacement de cette médiatrice.

Il peut être aussi intéressant de tester la méthode de hierarchie ascendante avec la méthodes des moments d'ordre 2, puis de prendre les centres des régions obtenus pour lancer un Kmeans.

Veysseire Daniel TP1 IAA Université Paul Sabatier

Contact

Si il y a des erreurs, des remarques, des ajouts à faire, etc.
Veuillez m'en faire part à cette adresse :

wedg@hotmail.fr