ENSTA PARIS

MIV308 : RECONNAISSANCE DE FORMES

TDs : K-PPV & K-Means & CAH

Réalisé par :
Amin HENTETI

Enseignant :
M. Taha RIDENE
M. IENG Sio-Song

Année universitaire: 2019/2020

1 TD N°1 – K-PPV

1.1 Principe algorithme *K-PPV*

L'algorithme *K-PPV* se base principalement sur une boucle principale qui parcourt tous les points à classé. On calcule la distance entre le point considéré et les points déjà classés.Puis on cherche les k points les plus proches ; avec k est un paramètre d'entrée. Enfin on associe à ce point considère la classe la plus représentée parmi ces points les plus proches.

1.2 Implémentation algorithme *K-PPV*

Mon implémentation de cet algorithme appliqué à la carte donné en énoncé, est codé par Matlab.
La structure des données d'entrée est considérée comme étant une matrice ses lignes correspondent aux différents points dans la carte. Chaque points est caractérisée par les donnés suivants : les coordonnées x et y, numéro et le classe alors j'associe à chaque donné une colonne respectivement ; ce qui signifie ;

- 1er colonne : coordonné x des points
- 2éme colonne : cordonnées y des points
- 3éme colonne : numéro du points dans la carte.
- 4éme colonne : numéro de la classe à laquelle la point appartient

Cette structure est représentée dans la matrice **N** qui contient les points à classer. Pour les autres qui sont déjà classés, j'ajoute des 4e colonne qui représentent le numéro de classe avec :

- numéro 1 référence la classe A de l'énoncé.
- numéro 2 référence la classe B de l'énoncé.

Le code contient plusieurs commentaires qui explique en détail chaque ligne de code et les variables employées. La structure du code est en premier lieu définir les données sous forme de matrice comme mentionné ci-dessus. En deuxième lieu, appliquer l'algorithme *K-PPV* pour classifier tous les points comme expliqués dans la partie précédente. Enfin afficher le résultat de classification comme montré dans la figure 1.

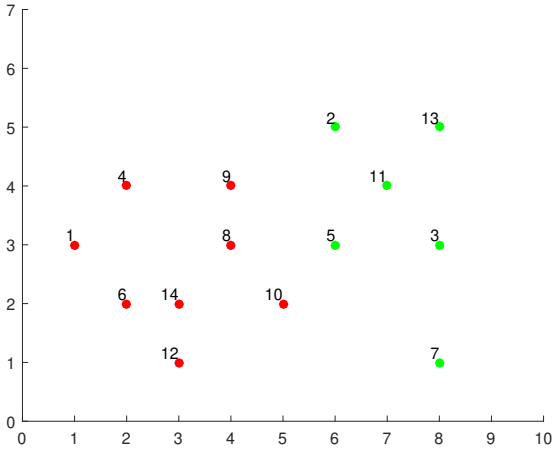


Figure 1: Résultat classification avec algorithme *K-PPV* : couleur rouge pour classe A et couleur vert pour classe B

2 TD N°2 – K-Means

2.1 Principe algorithme *K-Means*

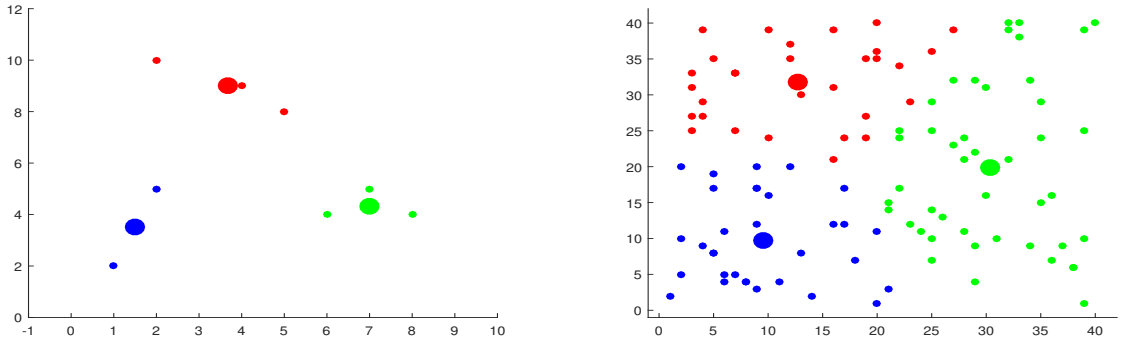
L'algorithme *K-Means* qui se base à deux boucles **for** imbriqué dans une **while**. Dans la première boucle **for** on associe à chaque point la classe du centre le plus proche et dans la deuxième **for** on fait la mise à jour des centres de chaque classe en le considérant comme étant le centre de gravité des nouveaux membre de chaque classe issue de la première boucle **for**. La boucle **while** se termine lorsque les centre presque ne varient pas.

2.2 Implémentation algorithme *K-Means*

Mon implémentation par Matlab de cet algorithme est appliqué à la carte donnée en énoncé (voir figure 2(a)). Je lui applique aussi pour des donnés générés de manière aléatoire (voir figure 2(b)).

La structure des données d'entrée représentée dans la matrice **A** est pareille que dans l'algorithme *K-PPV* qui contient les points à classer. On peut supprimer la 3e colonne : numéro du point dans la carte qui est facultative mais en defaux on doit bien préciser les centres c'est-à-dire la variable **Mu**.

Le code est bien commenté pour expliquer en détail chaque ligne de code et les variables employées. La structure du code est pareille que l'algorithme *K-PPV*. On commence par définir les données sous forme de matrice. Puis on appliquer l'algorithme *K-Means* pour classifier tous les points comme expliqués dans la partie précédente. Enfin on affiche le résultat de classification comme montré dans la figure 1. Les différentes couleurs sont employées pour différencier les différentes classes.



(a) points aléatoires

(b) points aléatoires

Figure 2: Résultat classification avec algorithme *K-Means*

3 TD N°3 – CAH

Un petit commentaire sur la correction donné, les coordonnés du point G9 sont (2,2.2) au lieu de (2,2).

3.1 Principe algorithme *CAH*

La classification automatique Classification Ascendante Hiérarchique *CAH* se base sur l'écart de Ward, définie par :

$$d\left(\Gamma_m, \Gamma_l\right) := \frac{p_m p_l}{p_m + p_l} d_2\left(G_m, G_l\right)^2$$

Avec p_m et p_l sont les poids des deux classes Γ_m et Γ_l respectivement et G_m et G_l leurs centre de gravité associés. d_2 mesure la distance euclidienne entre deux points.
L'algorithme *CAH* consiste à associer à chaque nuage de points une matrice symétrique

$$\mathbb{W} = (W_{ij})_{0 \leq i \leq n, 0 \leq j \leq n} = \left(d\left(M_i, M_j\right)\right)$$

Ensuite on cherche les classes (il peut contenir un seul point) qui donne le minimum non nul dans cette matrice. Puis on remplace ces deux classes par une nouvelle classe caractérisée par le centre de gravité de ces deux classes et de poids égal à leurs sommes. Les classes dans la population diminuent progressivement. On répète cette procédure $n - 1$ avec n est la taille de la population initiale.

3.2 Implémentation algorithme *CAH*

Mon implémentation de cet algorithme appliqué à la carte donné en énoncé, est codé par Matlab.
La structure des données d'entrée représentée dans la matrice **A** est constituée de deux colonnes : les coordonnées de chaque point à classer.

Le code contient plusieurs commentaires qui explique en détail chaque ligne de code et les variables employées. La structure du code est comme les deux précédents algorithmes. Un affichage du résultat de classification est montré dans la figure 3.

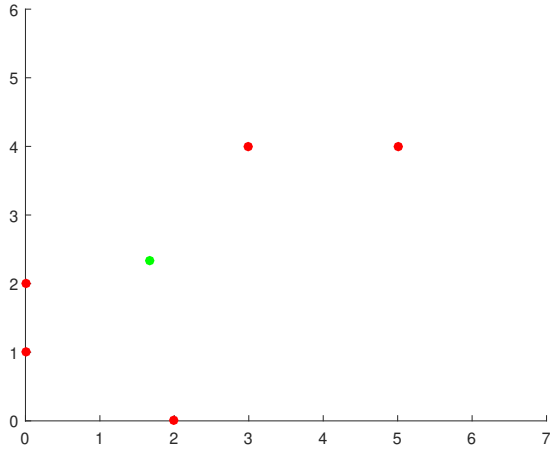


Figure 3: Résultat classification avec algorithme *CAH*