Rapport TP Clustering

DAHMANE

DJAMEL

Dans ce TP nous allons réaliser un Algorithme qui permet de classifier des points générées aléatoirement dans un espace 2D dans N cluster (N sera défini a l'avance) selon la distance entre le point et les centres des Cluster .

Pour cela nous allons définir une structure qui permettra de représenter un point avec deux attributs X et Y et une méthode calculerDistance qui permettra de calculer la distance entre le point en question et un autre point passe en paramètre.

Représentation de l'ensemble :

```
class Ensemble
    {
    private:
        /*  On sauvegarde les points de l ensemble dans deux vecteurs X et Y */
        std::vector<double> xarray;
        std::vector<double> yarray;


    public:
        Ensemble(std::vector<RandomGenerator> const  & generators,int nombrePoints);
        void addPoint(Point point);
        Point getPoint(int position);
        int getSize();
        std::vector<Cluster> getRandomCentroids(int number);
        int kMeansClustering (int epochs, int k,char animation);
};
```

Pour l'ensemble de points on vas utiliser une classe Ensemble qui contient deux vecteurs de points xarray et yarray.

Le constructeur prendra en paramètres un vecteur de RandomGenerator et nombrePoints pour construire N points aléatoires et les sauvegarder dans xarray et yarray ;

RandomGenerator :

```
class RandomGenerator {
    std::random_device r;
    std::default_random_engine e1;
    std::uniform_real_distribution<double> randomX;
    std::uniform_real_distribution<double> randomY;

    public:
    RandomGenerator(const RandomGenerator&);
    RandomGenerator(double Xmin, double Xmax,double Ymin, double Ymax);
    double getRandomX();
    double getRandomY();
    Point getPoint();
};
```

La classe RandomGenerator permet de générer des points aléatoires compris dans l'intervalle spécifié dans le constructeur

$$Xmin \leq X \leq Xmax \quad et \quad Ymin \leq Y \leq Ymax$$

Cluster

```
class Cluster {

private:
    static int id ;
    int clusterId;
    Point center ;
    std::vector<double> xarray;
    std::vector<double> yarray;
public:
    Cluster(Point p);
    void addPoint(Point point);
    double getDistance(Point p );
    void setCenter(double x,double y);
    std::vector<double> getXArray();
    std::vector<double> getYArray();

    void calculateCenter();
    double getCenterX();
    double getCenterY();
    int getClusterId();
    void reset();
    Point getCenter();
};
```

La classe Cluster a un constructeur qui prend en paramètre le Centroid initial choisi par défaut dans l'algorithme de Kmeans , et le clusterId est attribué a travers la variable statique id qui sera incrémente a chaque création d'un cluster .

La fonction calculateCenter() permet de calculer le centre d'un cluster .

La fonction reset() permet de vider le cluster pour une nouvelle itération dans l'algorithme de Kmenas

L'Algorithme Kmeans est implémenté dans la classe Ensemble

```
int Ensemble::kMeansClustering(int epochs, int k,char animation)
```

et prend 3 variables en paramètres : epochs : nombre d'iteration maximale

k : nombre de cluster attendue

animation : pour spécifier si l'utilisateur veut voir l'évolution de clustering dans chaque itération ou bien voir le résultat seulement

La fonction Main :

```
int main() {

    int nbCluster;
    int nbPoints;
    char  animation;
    cout<<"How many cluster do you want as result ? :  ";
    cin >> nbCluster;
    cout<<"How many points do you want ? : ";
    cin>>nbPoints;
    cout<<"Finally , do you want to view the clustering phases with an animation  ? y/n";
    cin >> animation;

    RandomGenerator r1(0,100,0,100);
    RandomGenerator r3(500,700,0,100);
    RandomGenerator r4(500,700,400,800);
    RandomGenerator r2(150,300,120,400);
    vector<RandomGenerator> listeGenerator;
    listeGenerator.push_back(r1);
    listeGenerator.push_back(r2);
    listeGenerator.push_back(r3);
    listeGenerator.push_back(r4);
    Ensemble ensemble(listeGenerator,nbPoints);
    ensemble.kMeansClustering(20,nbCluster,animation);
    return 0;

}
```

Dans la fonction Main on demande a l'utilisateur de saisir le nombre de Cluster résultant et aussi le nombre de points générées aléatoirement et aussi on demande a l'utilisateur si il veut voir l'animation des étapes de clustering (y : pour oui , autre pour nn)

on initialise 4 RandomGenerator et on les passe en paramètre pour l'ensemble crée et on lance l'algorithme .

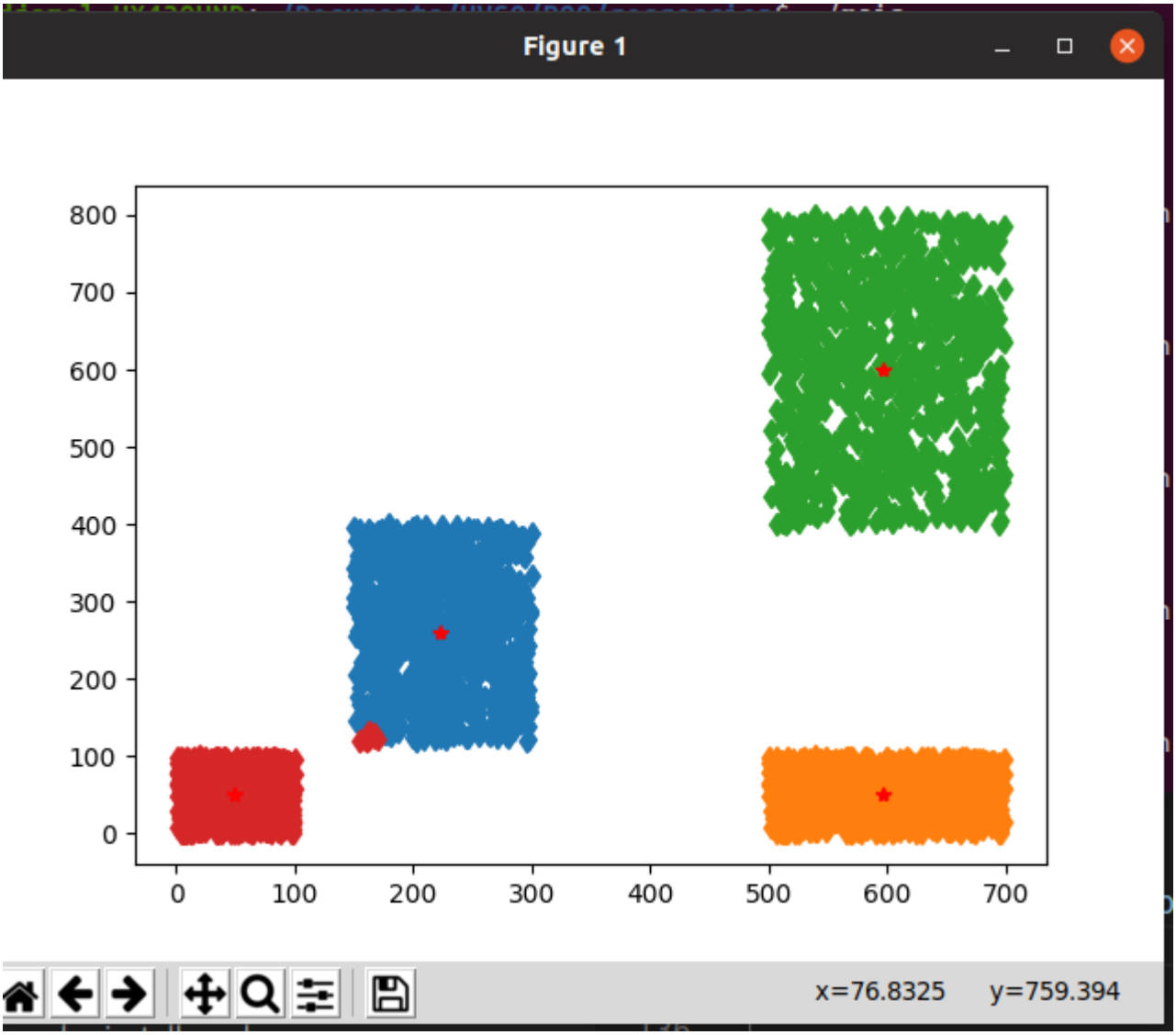
Résultat :

On lance le programme avec ces entrées :

```
djame@djamel-UX430UNR:~/Documents/UVSQ/POO/regression$ ./main
How many cluster do you want as result ? :  4
How many points do you want ? : 900
Finally , do you want to view the clustering phases with an animation  ? y/nn
```

Donc 4 clusters et 900 points sans animation .

Et voici le résultat



On a 4 clusters .