projet Kmeans

Hanene Azzag, Mustapha Lebbah, Dominique Bouthinon

1 Sujet

Le but du projet est d'implémenter en scala l'algorithme de clustering kmeans (kmoyennes) et de l'appliquer à la découverte de différentes catégories (classes) d'iris (`https://www.jardiner-malin.fr/fiche/iris.html`).

2 Les données

Les données utilisées pour tester votre programme sont les mesures en centimètres des caractères suivants : longueur du sépale (Sepal.Length), largeur du sépale (Sepal.Width), longueur du pétale (Petal.Length) et largeur du pétale (Petal.Width) de trois espèces d'iris : *Iris setosa*, *Iris versicolor* et *Iris virginica* correspondant aux 3 catégories possibles d'iris.

Le fichier *iris.data* contient les caractères de 150 iris ainsi que leur catégorie, le fichier *irisAttributes.names* contient les noms des attributs.

3 Les classes fournies

Vous utiliserez la programmation objets dans votre implémentation.
Les classes suivantes sont fournies avec leur documentation :

1. La classe *Individu* gère n caractéristiques (type Double) d'un individu (ici les caractères d'une iris) d'une population statistique.
2. La classe *Exemple* : un exemple est un *Individu* associé à un numéro de classe (Int) et un libellé de classe (String) (ici une iris, sa catégorie et un numéro de catégorie).
3. La classe *Data* gère une matrice d'exemples contenant les données initiales et une matrice contenant les données normalisées.

3.1 La classe *Data*

La classe *Data* permet de collecter un ensemble d'exemples en vue d'un kmeans. Dans le programme ci-dessous on remplit un objet *donnees* de type *Data* à partir des fichiers *iris.data* et *irisAttributes.names* :

```
object testKmeans
{
   def main(args : Array[String]) : Unit =
   {
               // on créé un objet donnees de type Data
               // qui collecte les données depuis les fichiers
               // iris.data et irisAttributes.names
      val donnees = new Data("iris.data", "irisAttributes.names")
   }
}
```

Les données doivent être automatiquement normalisées, les données initiales et les données normalisées cohabitant au sein du type *Data*

Si on veut afficher les données on ajoute l'instruction :

```
donnees.displayAllData()
```

qui produit l'affichage :

```
0 : (5,10;0,22) (3,50;0,62) (1,40;0,07) (0,20;0,04)  Iris-setosa (0)
1 : (4,90;0,17) (3,00;0,42) (1,40;0,07) (0,20;0,04)  Iris-setosa (0)
...
149 : (5,90;0,44) (3,00;0,42) (5,10;0,69) (1,80;0,71)  Iris-virginica (2)
```

qui affiche les 150 données. Les 4 couples de données *(valeur initiale ; valeur normalisée)* représentent les valeurs des 4 caractères *sepal length, sepal width, petal length, petal width.*

4 Les classes à écrire

4.1 La classe *Cluster*

La classe *Cluster* gère un cluster d'exemples. Dans le programme suivant on créé un cluster initialement vide associé aux exemples normalisés contenus dans un objet *donnees* de type *Data* :

```
object testKmeans
{
   def main(args : Array[String]) : Unit =
   {
               // on créé un objet donnees de type Data
      val donnees = new Data("iris.data", "irisAttributes.names")

               // recupere les exemples normalisés de donnees
      val exemples = donnees.getNormalizedData()

            // création d'un cluster initialement vide
            // de nom ''cluster1''
            // associé aux exemples normalisés
            // les  exemples sont décrits par 4 attributs,
            // et il y a 3 catégories d'exemples
      val cluster = new Cluster(''cluster1'', exemples, 4, 3)

            // ajoute dans le cluster les numeros de 5 exemples
            // présents dans exemples :
      cluster.add(4)
      cluster.add(6)
      cluster.add(123)
      cluster.add(19).
      cluster.add(77).

            // affiche le contenu du cluster
      println(cluster.toString)   // ou simplement println(cluster)

   }
}
```

4.2 La classe *Kmeans*

La classe *Kmeans* gère un clustering d'exemples stockés en interne dans une valeur de type *Data*. Dans le programme suivant on créé un objet *kmeans* de type *Kmeans* qui lit et stocke les données présentes dans les fichiers *iris.data* et *irisAttributes.names* :

```
object testKmeans
{
   def main(args : Array[String]) : Unit =
```

```
    {
                        // créé l'objet kmeans
        val kmeans = new Kmeans("iris.data", "irisAttributes.names")

                        // affiche les données présentes
        kmeans.displayAllData()
    }
}
```

La classe K-means doit déclarer une variable de type Data et y mettre les données (regarder la documentation et le contenu de la classe Data).

Votre classe K-means doit contenir une méthode *clustering(k : Int) : Double* qui effectue un K-means de k clusters et retourne la qualité du clustering. Regarder l'algorithme K-means donné en cours.

Rappel :

— distance euclidienne entre deux éléments $X = (x_1, ..., x_p)$ et $Y = (y_1, \cdots, y_p)$:

$$d(X, Y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_p - y_p)^2}$$

— centroid d'un cluster : soit un cluster ayant n éléments $X_1, \cdots, X_n$, où $X_i = (x_{i,1}, \ldots, x_{i,p})$, le centroid du cluster est de coordonnées :

$$C = (\frac{1}{n}\sum_{i=1}^{n} x_{i,1}, \cdots, \frac{1}{n}\sum_{i=1}^{n} x_{i,p})$$

— distance intra cluster : la distance intra-cluster est la moyenne des distances au carré des points au centroid C :

$$\text{dIntra} = \frac{1}{n}\sum_{i=1}^{n} d^2(X_i, C)$$

— distance inter clusters : Soit k clusters ayant les centroids $C_1, \cdots, C_k$ la distance inter-clusters est la moyenne des distances entre les centroids des clusters :

$$\text{dInter} = \frac{2}{(k-1) \times k}\sum_{i=1}^{k-1}\sum_{j=i+1}^{k} d(C_i, C_j)$$

la formule effectue la somme des distances :

$$\begin{array}{llll} d(X_1, X_2)+ & d(X_1, X_3)+ & \cdots & +d(X_1, X_k) \\ & +d(X_2, X_3)+ & \cdots & +\ d(X_2, X_k) \\ \ldots & & & \\ & & & +d(X_{k-1}, X_k) \end{array}$$

et divise par le nombre de distances : $\frac{(k-1)\times k}{2}$.

— la qualité d'un clustering est

$$\text{qualite} = \frac{\text{dInter}}{\text{moyenne des distrances intra-cluster}}$$

Plus la qualité est grande, meilleur est le clustering

5 Experimentation

Pour un k donné, effectuer 20 clusterings et calculer la moyenne des qualités obtenus. Faites varier k de 2 à 10 et conserver dans un tableau la moyenne des qualités pour chaque k.

6 Organisation

le projet est à faire OBLIGATOIREMENT EN BINÔME (aucun trinôme, et aucun monôme n'est autorisé). Le travail individuel de chaque membre du binôme sera précisé et évalué, ainsi deux membres d'un binôme pourront avoir 2 notes différentes.

Le projet est à rendre par mail au plus tard le 31 Juillet 2022, aucun retard ne sera admis.

7 rendu

Le rendu du projet doit être sous format ZIP (pas d'autre format accepté) et doit contenir :

1. Les programmes scala.
2. Un exécutable scala directement utilisable (sans compilation) DANS UN TERMINAL sur les données iris (il ne faut pas avoir à ouvrir un IDE pour lancer le programme).
3. Un compte rendu détaillé des expériences effectuées sur les données Iris comprenant l'interprétation des résultats.
4. Un bilan technique de votre projet (ce qui fonctionne, ne fonctionne pas, etc...)
5. la description du travail de chaque membre du binôme (une évaluation orale du travail pourra être effectuée).

Ce projet étant noté :

— Les projets manifestement similaires seront très fortement sanctionnés (de façon identique pour le copiant et le copié)
— Toute copie d'un code k-means pris sur internet sera très fortement sanctionnée.
— Tout projet rendu en retard ne sera pas considéré