UNIVERSITÉ TOULOUSE III PAUL SABATIER — Université de Toulouse

TP 1 : algorithme des k-moyennes

M1 IAA

Objectifs

1. Coder l'algorithme k-means en Python
2. Le tester sur des données numériques artificielles et des données réelles (fleurs iris)

Consignes :

1. nous allons utiliser Python 3 et numpy
2. il est recommandé d'utiliser un IDE pour Python comme PyCharm ou Spyder, ainsi que IPython comme interpréteur interactif

1 Implémentation de l'algorithme k-means

Algorithme k moyennes

1: Choisir au hasard k points parmi les données qui représentent la position des centroïdes initiaux
2: Répéter jusqu'à convergence :
3: a. assigner chaque observation à la partition la plus proche (i.e. effectuer une partition de Voronoï selon les moyennes) :

$$S_i^{(t)} = \left\{ \mathbf{x}_j : ||\mathbf{x}_j - \mathbf{m}_i^{(t)}|| \leq ||\mathbf{x}_j - \mathbf{m}_{i^*}^{(t)}||, \forall i^* \in [1, k] \right\}$$

4: b. mettre à jour la moyenne de chaque cluster (nuage) :

$$\mathbf{m}_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j$$

Les fichiers suivants vous sont fournis, ceux que vous devez compléter sont indiqués par une étoile :

— `main_kmeans.py` : programme principal ;
— `kmeans.py*` : algorithme des k-moyennes en lui-même ;
— `utils.py` : charge les données, affichage, etc.

Ouvrez dans l'IDE les différents programmes fournis et lisez brièvement leur contenu.

Remarque : si vous utilisez IPython pour exécuter votre programme, tapez les deux premières lignes suivantes (le faire une seule fois, ça suffit pour toute une session). Ces lignes magiques vous éviteront de devoir recharger les fonctions après chaque modification :

```
%load_ext autoreload
%autoreload 2
```

Vous pouvez ensuite exécuter ce programme dans IPython en tapant `run main_kmeans`, ou alors directement dans votre IDE à l'aide du bouton en forme de flèche verte.
Le programme ne fonctionne pas correctement car vous allez devoir compléter les programmes.

A FAIRE : Complétez les méthodes suivantes de la classe KMeans du fichier `kmeans.py` dans l'ordre. Chaque fonction devra être testée séparément avant de passer à la suite.

1. `_compute_distance(vec1, vec2)` : retourne la distance euclidienne au carré entre les vecteurs vec1 et vec2 de dimension quelconque ;
2. `_compute_inertia(X, y)` : retourne la somme des distances quadratiques entre les points et le centre de leur cluster associé. C'est une mesure "d'erreur" qui doit diminuer avec les itérations de votre algorithme. `y` est la liste des prédictions contenant le numéro du cluster attribué à chaque point ;
3. `predict(X)` : cette fonction associe à chaque point de `X` l'indice du centre qui lui est le plus proche. Elle retourne la liste `y` qui est la liste de ces indices ;
4. `_update_centers(X, y)` : cette fonction recalcule les coordonnées des centres des clusters. Elle met à jour l'attribut de classe `cluster_centers` qui est un tableau numpy contenant sur chaque ligne les coordonnées d'un centroide. `cluster_centers[i][j]` permet d'accéder à la j^{eme} coordonnée du i^{eme} centre ;
5. `fit(...)`, rajouter une actualisation du booléen `stabilise` dans la boucle pour que l'algorithme s'arrête lorsque la différence **absolue ou relative** de SSE entre deux itérations est inférieure à la tolérance `tol` ;
6. `fit(...)` : changer l'initialisation des centres pour qu'elle soit aléatoire (parmi les points des données toujours). Vérifiez que cette initialisation aléatoire des centroïdes a un effet sur les nuages de points que l'on obtient.

Questions

1. Exécuter l'algorithme en variant le nombre de centres (l'affichage fonctionne seulement pour un nombre de centres inférieur ou égale à 12),
2. L'algorithme tel qu'il a été implémenté ici est-il adapté au jeu de données fourni ? Que faudrait-il faire pour réaliser un partition des données satisfaisante ?

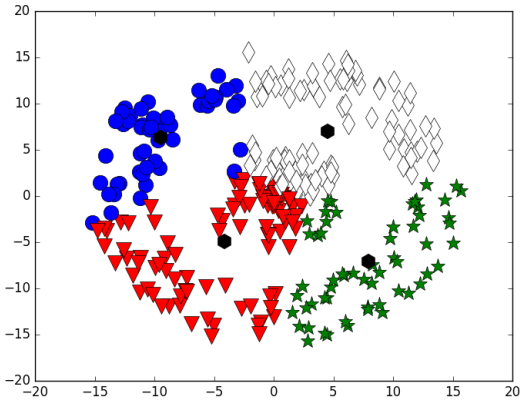

FIGURE 1 – Clusters obtenus sur les données numériques avec K=4. Vous devez obtenir une figure similaire.

Vous devez obtenir une figure similaire à la figure 1 si vous utilisez un nombre de 4 clusters.

Notes :

1. la classe `KMeans` contient un attribut `cluster_centers` contenant toutes les coordonnées des centroides ;
2. Les centroides sont affichés par leut numéro du cluster lors de l'affichage et non par un simple point.

2 Test sur les données Iris

Exécuter le programme sur les données "iris". Il s'agit de données sur 3 types de fleurs iris différentes : Iris-setosa, Iris-versicolor et Iris-virginica.

FIGURE 2 – Différents types de fleurs

Vous devez obtenir une figure similaire à la figure suivante lorsque vous prenez les deux dernières features pour abscisse et ordonnée respectivement (modifier la fonction d'affichage pour cela).

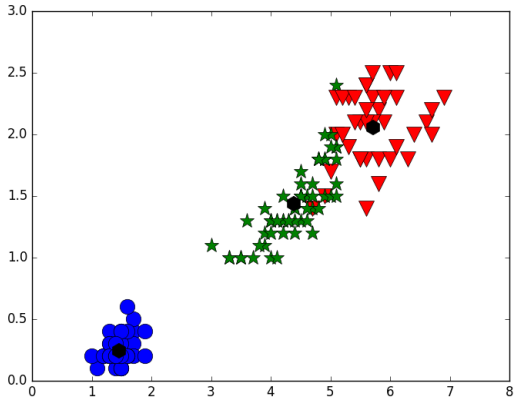

FIGURE 3 – Clusters obtenus sur les données iris