TP

L'algorithme des k-moyennes

Rappelons que l'algorithme des k-moyennes (*k-means* en anglais) consiste, à partir d'un ensemble de données, à répartir ces dernières en k classes formant une partition de l'ensemble en cherchant à minimiser l'inertie totale de l'ensemble (cette notion sera redéfinie plus loin dans le sujet).
Dans un premier temps, nous allons programmer cet algorithme, avant de s'intéresser à la meilleure façon de choisir le nombre k de classes à réaliser.

Préambule

Commencez par récupérer sur le site de la classe, à l'adresse

`https://pc-etoile.schola.fr/travaux-pratiques-dinformatique/`

le fichier `kmeans_data.py`. Ce dernier contient deux ensembles de points, `data1` et `data2`, qui nous servirons d'exemples. Ces ensembles de points sont représentés par des listes de points, chaque point étant lui-même représenté par une liste à deux éléments, son abscisse et son ordonnée.
Ainsi, `data1[i]` représente le $(i+1)^e$ point du premier ensemble, `data1[i][0]` son abscisse et `data1[i][1]` son ordonnée.

Le fichier `kmeans_data.py` contient aussi la définition d'une fonction `affiche` qui prend indifféremment pour argument une liste de points ou une liste de listes de points formant une partition de l'ensemble. Dans le premier cas, la fonction affiche à l'écran tous les points en une même couleur ; dans le second cas, chaque groupe formant la partition sera affiché avec une couleur différente.
Ainsi, vous pouvez désormais visualiser le premier ensemble de points en écrivant : `affiche(data1)`.

1. L'algorithme des k-moyennes

Question 1. Rédiger une fonction `barycentre(data)` qui prend pour argument un ensemble de points et renvoie les coordonnées du barycentre de ces points.

```
In [1]: barycentre(data1)
Out[1]: [4.1897208552738725, 4.619834096332582]
```

Question 2. Rédiger une fonction `dist(p, q)` qui prend pour arguments deux points p et q et renvoie $\mathrm{d}(p,q)$, le carré de la distance euclidienne de p à q.

```
In [2]: dist(data1[42], data1[24])
Out[2]: 55.50182505173223
```

Question 3. Rédiger une fonction `plusProche(p, mu)` qui prend pour arguments un point p et un ensemble de points $\mu = (\mu_1, \dots, \mu_k)$ et renvoie l'indice $i \in [\![0, k-1]\!]$ correspondant au point μ_i le plus proche de p.

```
In [3]: plusProche([0, 0], data1)
Out[3]: 54
```

Rappelons maintenant le principe de l'algorithme des k-moyennes :

(1) on choisit aléatoirement k points $\mu_1, \dots, \mu_k$;

(2) l'ensemble des données est partitionné en k *clusters*, chacun des points d'un même cluster étant associé au centre μ_i le plus proche ;

(3) on calcule les barycentres $\mu_1, \dots, \mu_k$ de ces clusters, qui remplacent les valeurs précédentes ;

(4) on reprend le calcul à partir de l'étape (2).

L'algorithme se termine lorsque les barycentres ne sont plus modifiés lors de l'étape (3).

Pour éviter d'obtenir des clusters vides lors de la première étape (ce qui provoquerait une erreur lors du calcul du barycentre), on prendra pour valeurs initiales de $\mu_1, \ldots, \mu_k$ lors de l'étape (1), k points distincts parmi les données à traiter (ce qui garantit que chaque cluster contient au minimum un point). Ceci sera réalisé par l'expression :

```
mu = sample(data, k)
```

(la fonction `sample(data, k)` du module `random` renvoie k valeurs distinctes prises dans la liste `data`).

Question 4. Rédiger une fonction `kmeans(data, k)` qui prend pour arguments un ensemble de points et un entier et qui renvoie la partition en k clusters de cet ensemble suivant l'algorithme des k-moyennes.
Cette partition sera représentée par une liste de k listes de points, ce qui en permettra la visualisation à l'aide de la fonction `affiche`.

```
In [4]: clusters = kmeans(data1, 4)
In [5]: affiche(clusters)
```

Une amélioration de l'algorithme

L'un des objectifs de l'algorithme des k-moyennes est de minimiser l'*inertie* de la partition $(C_1, \ldots, C_k)$ obtenue, autrement dit minimiser la quantité :

$$m = \sum_{i=1}^{k} \sum_{p \in C_j} d(p, \mu_j)$$

où μ_j désigne le barycentre du cluster C_j.

Question 5. Rédiger une fonction `inertie(clusters)` qui prend pour argument une liste de clusters et qui renvoie l'inertie de ce partitionnement.

```
In [6]: inertie([data1, data2])
Out[6]: 5268.032211202934
```

On sait que l'algorithme des k-moyennes fournit un partitionnement qui correspond à un minimum local de l'inertie, mais pas forcément au minimum global. Pour augmenter nos chances d'obtenir ce minimum global, nous allons appliquer 20 fois de suite l'algorithme des k-moyennes pour ne garder que la réponse d'inertie minimale.

Question 6. Rédiger la fonction `bestKmeans(data, k)` qui renvoie le meilleur de ces 20 partitionnements.

```
In [7]: clusters = bestKmeans(data1, 4)
In [8]: affiche(clusters)
```

2. Le choix de l'entier k

Observer maintenant le clustering de l'ensemble de données `data2` pour $k = 3$ et pour $k = 4$. Lequel est le meilleur ? La réponse n'est pas évidente.
Pour le savoir, nous allons définir la notion de *silhouette* d'un point, puis d'une partition.

Question 7. Rédiger une fonction `moyenne(lst)` qui renvoie la moyenne des nombres contenus dans la liste (supposée non vide) `lst`.

Pour définir la *silhouette* d'un point p appartenant au cluster C_i d'une partition $(C_1, \ldots, C_k)$, on introduit successivement les quantités suivantes :

– $a(p)$ est la moyenne des distances entre p et les points q appartenant au même cluster que p :

$$a(p) = \frac{1}{\text{card}(C_i) - 1} \sum_{q \in C_i \setminus \{p\}} d(p, q)$$

– $b(p)$ est la moyenne des distances entre p et les points q du cluster le plus proche :

$$b(p) = \min_{j \neq i} \frac{1}{\mathrm{card}(\mathrm{C}_j)} \sum_{q \in \mathrm{C}_j} \mathrm{d}(p,q)$$

– la *silhouette* de p est alors la quantité $\mathrm{sil}(p) = \dfrac{b(p)-a(p)}{\max\big(a(p),b(p)\big)}$.

Par construction la silhouette de p est un réel compris entre 1 et -1. Lorsque $\mathrm{sil}(p)$ est proche de 1, cela signifie que $a(p) \ll b(p)$. Or une petite valeur de $a(p)$ signifie que p est bien intégré dans son cluster et une grande valeur de $b(p)$ que p ne s'intègre pas bien aux autres clusters. Bref, une valeur proche de 1 montre que le cluster associé à p est bien choisi. À l'inverse, une valeur de $\mathrm{sil}(p)$ proche de -1 signifie que $b(p) \ll a(p)$ et donc que le cluster attribué à p n'est pas pertinent.

Question 8. Rédiger successivement les fonctions `a(p, clusters)`, `b(p, clusters)` et `sil(p, clusters)` qui prennent pour argument un point et une partition et qui permettent de calculer la silhouette d'un point associé à une partition.

La *silhouette* d'une partition $(\mathrm{C}_1,\dots,\mathrm{C}_k)$ consiste à calculer la moyenne des silhouettes des points de chaque cluster, puis à faire la moyenne de ces moyennes :

$$\mathrm{sil}(\mathrm{C}) = \frac{1}{k}\sum_{i=1}^{k}\left(\frac{1}{\mathrm{card}(\mathrm{C}_i)}\sum_{p\in\mathrm{C}_i}\mathrm{sil}(p)\right)$$

Question 9. Rédiger une fonction `silhouette(clusters)` qui calcule la silhouette d'une partition.

```
In [9]: silhouette(bestKmeans(data1, 4)), silhouette(bestKmeans(data1, 5))
Out[9]: (0.8597810467277319, 0.7430478689324185)
```

On constate que le partitionnement de `data1` en quatre classes est plus pertinent que le partitionnement en cinq classes.

Pour combien de classes le partitionnement de `data2` est-il le plus pertinent ?