K-means

1 Objectif

L'objectif consiste à implémenter l'algorithme des K-means (ou tout du moins l'une de ses implémentations les plus connues, l'algorithme de Lloyd) :

- en "pur" Python
- avec NumPy
- en "base" R

L'utilisation d'un *line profiler* permettra de quantifier les gains envisageables permis par l'utilisation d'un "sous-langage" spécialisé pour le calcul scientifique comme `NumPy` par rapport à une implémentation en "pur" Python.

Pour référence, l'implémentation déjà existante de scikit-learn (fortement recommmandée pour tout usage un minimum sérieux) sera évoquée.

Sera également proposée une implémentation des K-means en "base" R, c'est-à-dire sans package supplémentaire. Cet exemple permettra de mettre en évidence la forte proximité qui peut exister entre les deux langages (en particulier avec la version NumPy).

2 Génération des données

La commande suivante permet de générer un jeu de données avec le nombre de points et de clusters souhaités :

```
python 00-generate_data.py --n_points <int> --n_clusters <int>
```

Les deux arguments `--n_points` et `--n_clusters` sont optionnels avec, respectivement, des valeurs par défaut de 150 et 3.

Les données ainsi créées sont enregistrées dans le sous-répertoire `data` sous la forme : `data_<n_points>.csv`

3 Implémentation des K-means en "pur" Python

Pour exécuter l'algorithme des K-means écrit en "pur" Python, il suffit simplement d'écrire :

```
python 01-kmeans.py data/<file>
```

Deux paramètres optionnels peuvent être précisés :

- le nombre de clusters avec l'argument `--n_clusters` (3 par défaut) ;

- le fait de créer un graphique pour chacune des étapes de l'algorithme avec le flag `--plot` (ou `-p`).

Par exemple, pour créer 4 clusters et activer la représentation graphique :

```
python 01-kmeans.py data/<file> --n_clusters 4 -p
```

Les figures générées seront stockées dans le sous-répertoire `plots`.

Pour convertir l'ensemble des figures en une animation, utiliser la commande suivante (les paramètres peuvent être ajustés à convenance) :

```
convert -delay 100 -loop 0 $(ls -1v plots/*.png) plots/<file>.gif
```

NB : il est nécessaire d'installer ImageMagick au préalable.

L'utilisation du module `line_profiler` (et de son script `kernprof`) rend le profiling de l'algorithme trivial ou presque :

- ajouter le *decorator* `@profile` devant la fonction a profilé (kmeans() dans le cas présent) ;
- exécuter la commande suivante :

```
kernprof -l -v 01-kmeans.py data/<file>
```

4 Implémentation des K-means avec NumPy

L'exécution du script peut se faire avec la commande :

```
python 02-kmeans_numpy.py data/<file>
```

Tout comme précédemment, le nombre de clusters peut être modifié avec l'argument `--n_clusters` (3 par défaut).

Pour profiler l'implémentation avec NumPy (après avoir ajouté le *decorator* `@profile` devant la fonction kmeans()):

```
kernprof -l -v 02-kmeans_numpy.py data/<file>
```

5 Implémentation des K-means avec scikit-learn

Pour exécuter l'implémentation des K-means de scikit-learn :

```
python 03-kmeans_sklearn.py data/<file>
```

Comme pour les trois implémentations précédentes, le nombre de clusters peut être ajusté avec l'argument `--n_clusters` (3 par défaut).

> *Dans cette implémentation, l'algorithme est – par défaut – répété 10 fois avec des centres initiaux différents. Le nombre d'itération maximum pour chaque répétition est limité à 300.*

6 Implémentation des K-means en "base" R

L'implémentation en R s'exécute de façon très similaire :

```
Rscript 04-kmeans.R data/<file>
```

Ici également, deux arguments optionnels sont proposés :

- le nombre de clusters avec l'argument `--n_clusters` (3 par défaut) ;
- le fait de créer un graphique pour chacune des étapes de l'algorithme avec le flag `--plot` (ou `-p`).

7 Arborescence

```
kmeans
|__ data
|   |__ ...
|__ plots
|   |__ ...
|__ 00-generate_data.py
|__ 01-kmeans.py
|__ 02-kmeans_numpy.py
|__ 03-kmeans_sklearn.py
|__ 04-kmeans.R
```