ML : K-Means

TP3 Apprentissage automatique

K-Moyennes

Jérôme Buisine
jerome.buisine@univ-littoral.fr

21 novembre 2022

Durée : 3h

L'objectif de ce TP est d'exploiter la librairie `scikit-learn` pour comprendre le fonctionnement de l'algorithme des K-moyennes.

1 Ressources

Voici un ensemble de ressources qui peuvent vous être utiles durant ce TP :

— 1. Documentation officielle de l'outil `Git` ;
— 2. Gitlab : interface web pour la gestion de projets versionnés sous `Git` ;
— 3. pyenv : permet la gestion d'environnement Python.
— 4. matplotlib : librairie Python qui permet un affichage rapide de données.
— 6. jupyter : un outil de travail permettant une interaction rapide et visuelle avec une console Python.

2 Configuration de l'environnement

Créez un dossier à la racine de votre projet nommé `tp3-k_means`. Dans ce dossier et depuis votre terminal, lancez les commandes suivantes :

```
# spécifie l'environnement virtuel Python à Jupyter
ipython kernel install --user --name=ml-venv
# lance l'application Jupyter
jupyter-lab
```

Remarque : pour chaque section suivante, il vous sera demandé de créer un notebook depuis Jupyter. Faites bien attention à la sélection de l'environnement Python lors de sa création. De plus, il vous sera demandé de bien documenter votre code. Vous pouvez à cet effet utiliser des cellules de type `Markdown` plutôt que de type `code`.

3 Compréhension des K-moyennes

Créez un nouveau notebook nommé « k_means_classical.ipynb ». Dans ce notebook nous allons traiter un cas simple de `clustering`.

3.1 Un premier exemple d'apprentissage

Nous allons tout d'abord générer des données à l'aide du module `sklearn.datasets`. Ces données nous servirons de base pour une bonne compréhension de la validation de modèle dans un cadre non-supervisé.

Tâche 1 : Créer un ensemble de données à partir de la fonction `make_blobs` du module `sklearn.datasets` tel que :

- → le nombre de `samples` soit de 1000.
- → le nombre de `centers` fixé à 3.
- → le nombre de `features` fixé à 10.
- → On peut également ajouté un `random_state` fixe, par exemple 42.

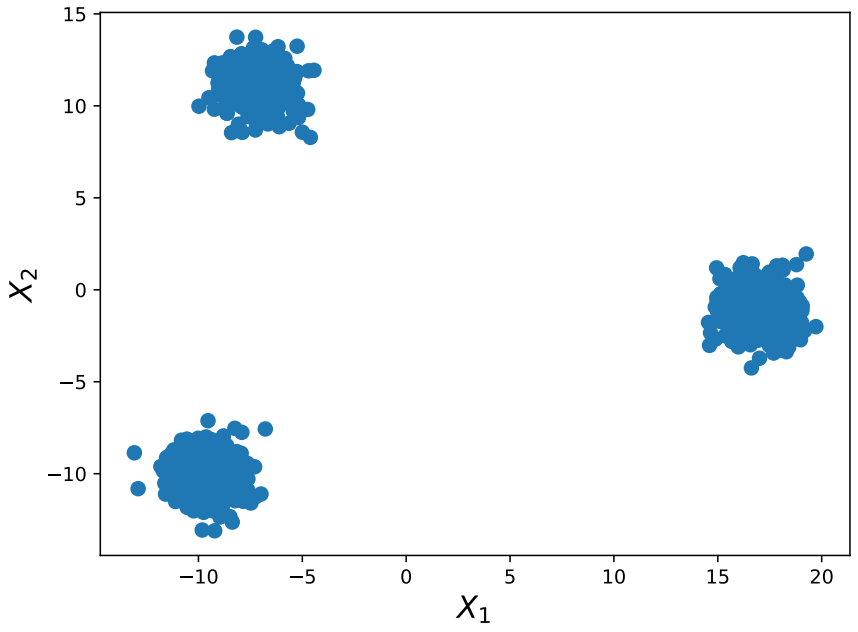
En l'état, étant donné que nous avons 10 features, il n'est pas possible de projeter nos données et de les visualiser.

Pour permettre un visuel de nos données, nous allons utiliser une méthode de réduction de dimensions. Ces méthodes sont disponibles dans le module `sklearn.decomposition`. Dans le cadre de ce TP, nous nous concentrons sur l'analyse en composantes principales (`PCA`). Cette méthode permet de conserver le maximum de variance dans un nombre réduit de nouveaux descripteurs (nommées composantes principales) dont le nombre est définit par l'utilisateur.

Tâche 2 : Créer un modèle de décomposition `PCA` tel que :

- → Le nombre de composantes souhaitées en sortie soit de 2.
- → Il faudra alors apprendre ce modèle à transformer nos données.
- → **Attention :** ce modèle ne prend que des données en entrée (voir documentation de la méthode `fit` de la méthode).

Voici un exemple d'affichage de données transformées en 2 composantes principales :



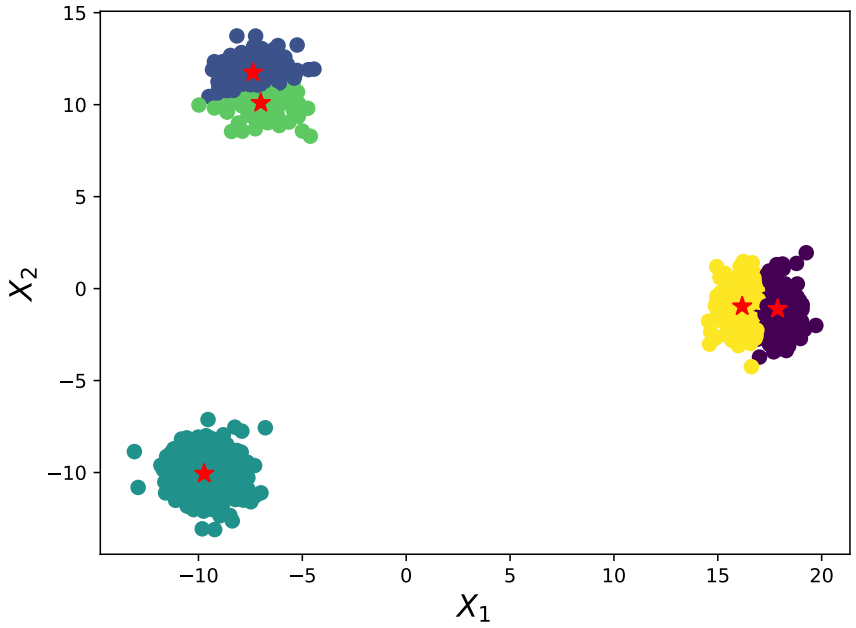
Remarque : on peut de suite visualiser les 3 clusters présents dans nos données. Toutefois, il faut garder à l'esprit que cela reste dans un cadre de compréhension et de validation du modèle. Cette séparation ne sera pas toujours si évidente.

Tâche 3 : Apprendre un modèle `sklearn.cluster.KMeans` avec un nombre de clusters fixé à 5 :

→ Apprendre le modèle sur les données réduites par la méthode PCA.

→ Il faudra accéder aux attributs du modèle ayant apprit pour associer des couleurs de classes à chaque point et afficher les centroïdes obtenus.

Pour $k = 5$, il est possible d'obtenir ce genre de classification non-supervisée :



3.2 Validation du paramètre k

L'exemple précédent montre bien que le choix de $k = 5$ n'est pas forcément le plus judicieux dans notre cas. Toutefois, il est important de le valider et l'expliquer par les performances du modèle.

Un critère de validation d'un modèle serait possible en calculant le carré des distances de chaque point par rapport au centroïde qui lui est associé. La formule est la suivante :

$$T = \frac{1}{m} \sum_{j=1}^{C} \sum_{i=1}^{m} \delta_i^j (\mathbf{x}_i - \mu_j)^2$$

avec :

- C le nombre de classes.
- m le nombre de points.
- μ_j le centroïde du cluster j.
- $\delta_i^j = 1$ si $i = j$ et 0 sinon.

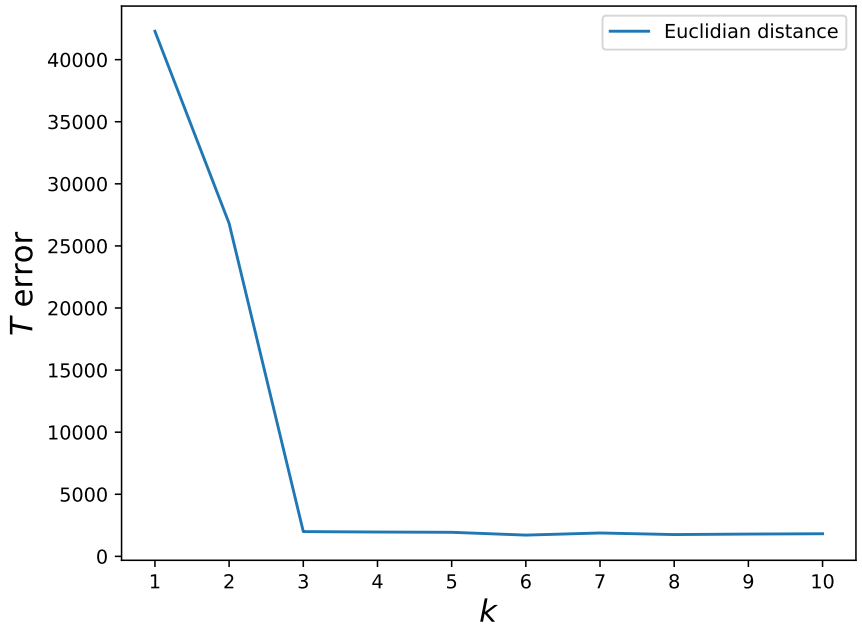
Tâche 4 : Écrire la fonction « `euclidian_distance_from_centroid` » qui calcule cette erreur, afin de :

→ Calculer les différentes erreurs pour $k \in [1, 10]$

→ Afficher les erreurs obtenues à l'aide de `matplotlib`. Il est nécessaire de mettre à jour les « `ticks` » en coordonnée x pour partir d'une valeur $k = 1$:

```
import numpy as np
kmax = 10
...
# ticks, labels
plt.xticks(np.arange(kmax), np.arange(kmax) + 1)
```

À partir de cette information, on peut de suite visualiser que $k = 3$ est effectivement bien un nombre de clusters attendu :



Une seule mesure n'est peut-être pas suffisante pour évaluer correctement si le nombre de clusters est cohérent. Il existe également la mesure *silhouette* $s(i)$ pour chaque cluster i. Elle est bornée par $-1 \leqslant s(i) \leqslant 1$ et définit telle que :

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \text{ if } |C_I| > 1$$

avec :

- $|C_I|$ le nombre de points pour le cluster C_I.
- $a(i) = \frac{1}{|C_I|-1} \sum_{j \in C_I, i \neq j} d(i,j)$, la moyenne des distances des points par rapport à leur centroïde respectif (pour chaque $i \in C_I$).
- $b(i) = \min_{J \neq I} \frac{1}{|C_J|} \sum_{j \in C_J} d(i,j)$, la plus petite distance moyenne de i à tous les points de tout autre cluster.

Finalement, le coefficient de *silhouette* pour k clusters est définit tel que :

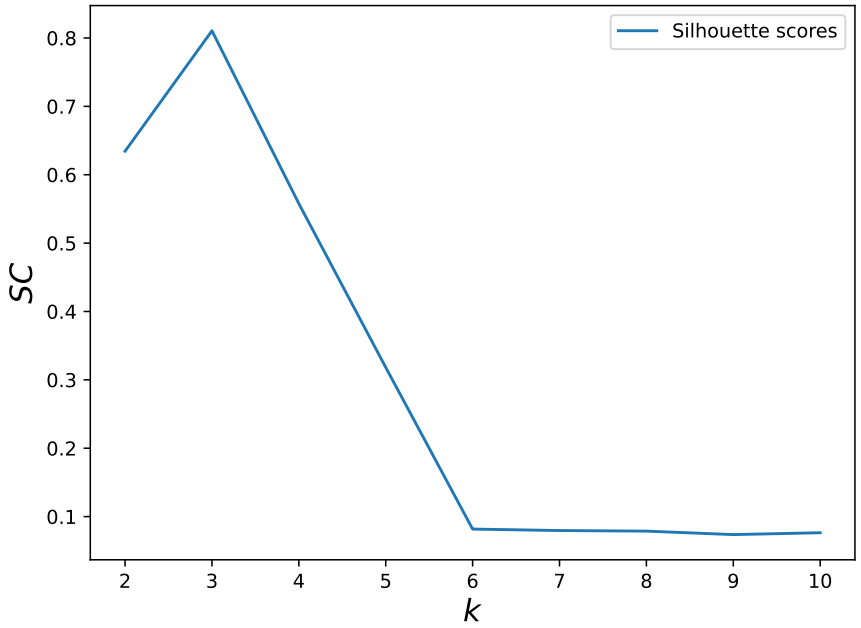
$$SC = \max_k \tilde{s}(k)$$

Tâche 5 : À partir du module `sklearn.metrics`, utiliser la fonction « `silhouette_score` » pour :

→ Calculer le score *silhouette* pour $k \in [2, 10]$ (cette mesure se base sur au moins 2 clusters).

→ Afficher les scores obtenues à l'aide de `matplotlib`.

À partir de la courbe obtenue, on peut remarquer qu'il ne semble pas judicieux d'augmenter le nombre de clusters au delà de $k = 6$:



Remarque : pour juger du choix du paramètre k, il peut être nécessaire de s'appuyer sur ces deux coefficients. Leur complémentarité peut s'avérer utile.

4 Application à un cas concret

Vous allez dans cette partie vous attaquez à un problème sur des données plus complexes. Créez un nouveau notebook nommé « k_means_use_case.ipynb ».

4.1 Validation d'un modèle sur des données plus difficilement séparables

Vous allez vous baser sur la base de données country qui propose des informations relatives à certains pays. L'objectif est de pouvoir judicieusement catégoriser ces pays à partir de leurs descripteurs.

Tâche 6 : À partir des connaissances acquises lors de la première partie de ce TP, utilisez l'algorithme des K-moyennes pour cibler le meilleur paramètre k :

→ La colonne *country* sera à négliger en entrée de l'algorithme.

→ Toutefois, cette colonne pourra permettre d'analyser les pays et leurs appartenances aux clusters.

À ce stade, vous devez avoir identifier un paramètre k qui vous semble cohérent.

4.2 Un potentiel besoin de mise à l'échelle

Nous avons actuellement négliger un paramètre important : la mise à l'échelle. Dans la manière de procéder, l'algorithme des K-moyennes ressemble à l'algorithme KNN qui lui, était sensible à cette mise à l'échelle. En effet, tous deux sont basés sur des notions de distance.

Tâche 7 : Explorer le besoin ou non de normaliser les données pour l'algorithme des K-moyennes :

→ Il est possible d'utiliser le `StandardScaler` vu avec le KNN, mais aussi le `MinMaxScaler`.

→ Il faudra identifier à quel niveau cette normalisation est nécessaire (avant analyse PCA ou post-analyse).

→ Évaluer les performances du modèle avec cette mise à l'échelle.

5 Résumé

- Les K-moyennes est un algorithme simple qui permet de catégoriser des données non labellisées.
- L'analyse en composante principale (PCA) permet de réduire en dimension les données et de donner une représentation simplifiée des clusters obtenus.
- Le choix du paramètre k peut être sélectionné/validé à l'aide de mesures de distance des points relativement aux centroïdes obtenus.
- La normalisation des données peut être importante pour une mise à l'échelle des données fournies de chaque descripteur.

6 Remise des travaux

N'oubliez pas de réaliser un commit de vos travaux. Taguez également votre projet avec le tag « `tp3` » et soumettez-le sur le serveur Gitlab. Il fera office de rendu.