Big Data. TD 3

Natalia Kharchenko, Sergey Kirgizov

ESIREM

k-means clustering

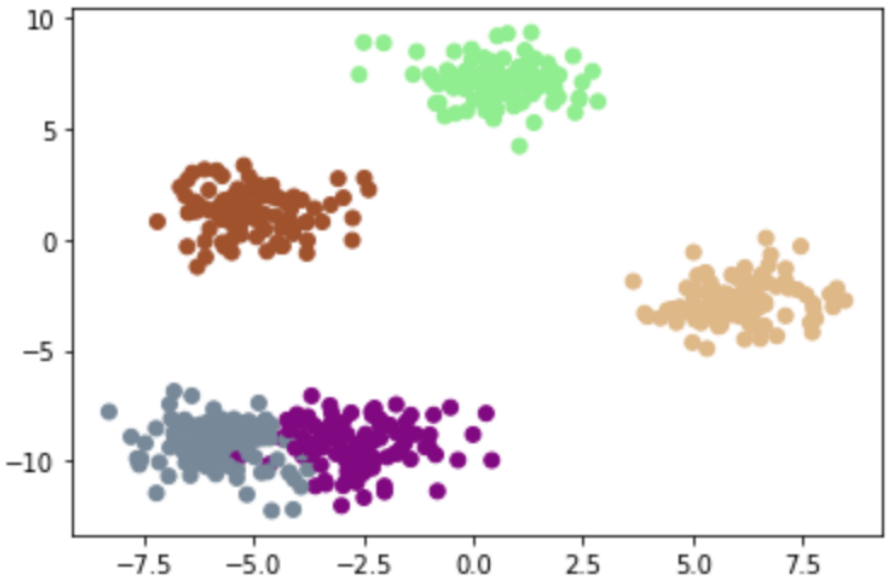
Le but de cet exercice est d'implémenter l'algorithme de clustering k-means et certaines de ses variations : k-means++ et la version mini-batch. Nous les testerons sur un petit ensemble de données artificielles et nous visualiserons les résultats pour avoir une idée de leur fonctionnement.

Technologies à utiliser : `python`, `numpy`, `sklearn`. *Veuillez utiliser* `jupyter notebook` *pour les tests et les visualisations et votre IDE préféré pour la partie codage.*

Préparation et visualisation des données

☞ **EXERCICE 1 :** Générer 5 clusters gaussiens qui contiennent 500 points bidimensionnels au total en utilisant la fonction `sklearn.dataset.make_blobs`.

☞ **EXERCICE 2 :** Créez une fonction `visualize_clusters` qui prend en entrée les points de cluster du même format que ceux retournés par `make_blobs` et dessine un nuage de points où chaque cluster est représenté par une couleur différente. Pour obtenir différentes couleurs, vous pouvez utiliser `matplotlib.colors.CSS4_COLORS` comme palette de couleurs. Voici un exemple du résultat.



L'algorithme basique de LLoyd

L'algorithme de base de Lloyd se compose des trois étapes suivantes.

1. *Initialisation des centres.* Dans la version la plus basique, k centres sont choisis au hasard parmi les points de données d'entrée.
2. *Mise à jour des clusters.* Chaque point est assigné au cluster dont le centre est le plus proche.

3. *Mise à jour des centres.* Chaque centre est remplacé par la moyenne des points de son cluster.

Les étapes 2 et 3 sont répétées jusqu'à convergence, c'est-à-dire jusqu'à ce que les centres ne changent plus.

EXERCICE 3 : Implémentez une fonction `initialize_centers` qui prend en entrée les points de données et le paramètre k et retourne k centres choisis au hasard parmi les points de données.

EXERCICE 4 : Implémentez une fonction `sse_distance` qui, étant donné deux points de données, calcule la distance euclidienne carrée entre eux.

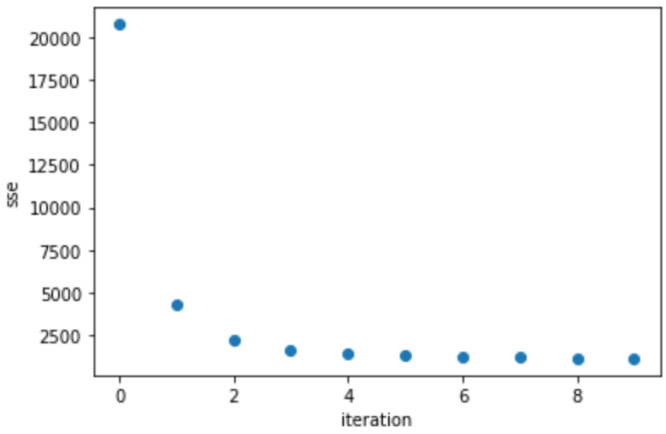
EXERCICE 5 : Implémentez une fonction `find_closest_center` qui, avec en entrée une liste de centres et un point de données, retourne l'index du centre le plus proche du point donné.

EXERCICE 6 : Implémentez une fonction `compute_clusters` qui, étant donné une liste de points de données et une liste de centres, retourne une liste de clusters correspondant à chaque point de données. Pour l'attribution des clusters, utilisez le même format que dans `sklearn.dataset.make_blobs`.

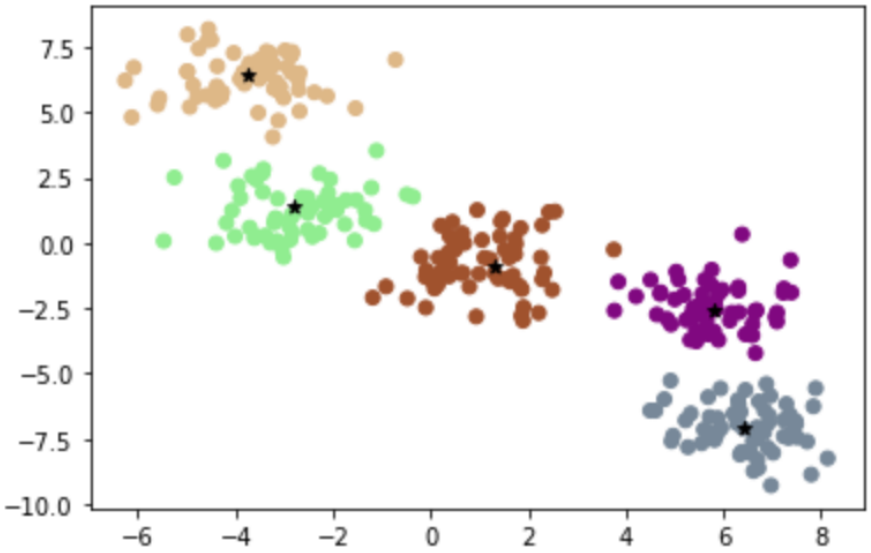
EXERCICE 7 : Implémentez une fonction `sse_error` qui prend en entrée une liste de points de données et une liste de centres et calcule la somme des carrés des distances euclidiennes de chaque point à son centre de cluster. Réutilisez les fonctions des exercices précédents.

EXERCICE 8 : Utilisez les fonctions des exercices précédents pour implémenter une fonction `kmeans`. Elle doit prendre en entrée la liste des points de données, le nombre de clusters k, et le nombre d'itérations `num_it` et exécute `num_it` itérations de l'algorithme k-means. La fonction doit créer un dict `history` qui stocke les k centres et `sse_error` à chaque itération.

EXERCICE 9 : Exécutez 10 itérations de l'algorithme en utilisant les données de `make_blobs` et créez un graphique pour voir comment SSE change avec les itérations. Normalement, vous obtiendrez quelque chose comme ça.



EXERCICE 10 : Changez la fonction `visualize_clusters` pour qu'elle prenne en entrée un argument supplémentaire : la liste des centres. Elle devrait les visualiser en plus de colorer les points de données. Le résultat peut ressembler à cela.



☞ **EXERCICE 11 :** En utilisant le dict `history` pré-enregistré et les fonctions `compute_clusters` et `visualize_clusters`, visualisez les données de chaque itération et regardez comment les clusters évoluent.

☞ **EXERCICE 12 :** Modifiez la fonction `kmeans` pour qu'elle puisse soit exécuter un nombre fixe d'itérations, soit exécuter jusqu'à convergence, c'est-à-dire jusqu'à ce que les centres n'évoluent plus. La fonction `numpy.close` pourrait être utile pour cet exercice. Exécutez l'algorithme jusqu'à convergence et notez le nombre d'itérations.

k-means++

L'algorithme k-means de base ne donne aucune garantie sur la qualité de la solution. Cependant, il existe une variation de l'échantillonnage initial qui permet de trouver une approximation de $O(\log k)$ de la solution optimale. Cette variation est appelée k-means++ et, outre les garanties de qualité théoriques, elle semble fournir des solutions plus stables en pratique.

☞ **EXERCICE 13 :** Apprenez comment l'échantillonnage initial fonctionne dans k-means++. Voir la section 2.2 du papier "k-means++ : The Advantages of Careful Seeding" de David Arthur, Sergei Vassilvitskii. `https://kirgizov.link/teaching/esirem/bigdata/kMeansPP-soda.pdf`

☞ **EXERCICE★ 14 :** Ajoutez une option d'échantillonnage k-means++ à votre algorithme kmeans.

☞ **EXERCICE★ 15 :** Comparez l'échantillonnage de base et kmeans++ en exécutant les algorithmes sur plusieurs ensembles de données aléatoires. Comparez le nombre d'itérations et la qualité de la solution finale.

Mini-batch (streaming) k-means

Il existe une version du kmeans qui permet de l'exécuter pour de grands ensembles de données qui ne tiennent pas dans la mémoire. Cette version est appelée mini-batch kmeans (ou streaming kmeans). Une itération de mini-batch kmeans utilise seulement un petit sous-ensemble de données pour mettre à jour les centres.

☞ **EXERCICE 16 :** Apprenez comment fonctionne le k-means mini-batch en lisant la section 2 et l'algorithme 1 du papier "Web-Scale K-Means Clustering" de D. Sculley. `https://kirgizov.link/teaching/esirem/bigdata/minibatch.pdf`. Pourriez-vous expliquer comment fonctionne le gradient step ?

☞ **EXERCICE★ 17 :** Implémenter le k-means mini-batch et le tester sur les données aléatoires comme les algorithmes précédents. Comparez les résultats avec les k-means de base et les k-means++.