CRTP1 IA et Données

INFO0503

07 novembre 2024

VERPOORTE William

Sommaire

1 Implémentation de KMeans

```python
import matplotlib.pyplot as plt
from math import sqrt
from typing import List
import random
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.metrics import silhouette_score


class Point:
    def __init__(self, x: float, y: float):
        self.x = x
        self.y = y

    def __repr__(self):
        return f"Point({self.x}, {self.y})"

def Kmeans(points: List[Point], centres: List[Point]):
    max_iterations = 10
    colors = ['red', 'blue', 'green', 'purple', 'orange', 'brown', 'pink', 'cyan', 'magenta', 'yellow']

    plt.figure(figsize=(8, 6))  # Cr er la fen tre pour le graphique
    for iteration in range(max_iterations):
        classe = {centre: [] for centre in centres}

        for pt in points:
            dist_min = float('inf')
            centre_min = None
            for centre in centres:
```

```
                dist = sqrt((pt.x - centre.x)**2 + (pt.y - centre.y)**2)
                if dist < dist_min:
                    dist_min = dist
                    centre_min = centre
            classe[centre_min].append(pt)

        # Effacer l'ancienne figure pour la mise   jour
        plt.clf()

        # Affichage des points associ s   chaque centre
        for idx, (centre, points_in_class) in enumerate(classe.items()):
            x_vals = [pt.x for pt in points_in_class]
            y_vals = [pt.y for pt in points_in_class]
            plt.scatter(x_vals, y_vals, c=colors[idx], label=f'Centre {centre} (Cluster {idx+1})', alpha=0.6)

        # Affichage des centres de cluster
        centre_x_vals = [centre.x for centre in centres]
        centre_y_vals = [centre.y for centre in centres]
        plt.scatter(centre_x_vals, centre_y_vals, c='black', marker='X', label='Centres', s=200)

        # Titre et l gende
        plt.title(f"K-means - Iteration {iteration + 1}")
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.legend(loc='upper right')
        plt.grid(True)

        new_centres = []
        for centre in centres:
```

```
            if classe[centre]:
                moyenne_x = sum(pt.x for pt in classe[centre]) / len(classe[centre])
                moyenne_y = sum(pt.y for pt in classe[centre]) / len(classe[centre])
                new_centres.append(Point(moyenne_x, moyenne_y))
            else:
                new_centres.append(centre)

        if all(abs(new.x - old.x) < 1e-4 and abs(new.y - old.y) < 1e-4 for new, old in zip(new_centres, centres)):
            print("Convergence atteinte!")
            break

        centres = new_centres

        plt.pause(3)  # Pause de 3 secondes

    plt.show()


# Exemple d'utilisation avec des points suppl mentaires
points = [Point(random.uniform(0, 10), random.uniform(0, 10)) for _ in range(100)]
points2 = [[random.uniform(0, 10), random.uniform(0, 10)] for _ in range(100)]


# Centres initiaux pour le k-means
centres = [
    Point(2, 2), Point(5, 5), Point(8, 3), Point(10, 10)
]

```

```
# Appel de la fonction kmeans avec la liste  tendue  de points
Kmeans(points, centres)

# Appliquer K-means avec 3 clusters
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(points2)

# Afficher les r sultats
plt.scatter([p[0] for p in points2], [p[1] for p in points2],
    c=kmeans.labels_)
plt.scatter(kmeans.cluster_centers_[:, 0],
    kmeans.cluster_centers_[:, 1], s=200, c='red', marker='X')
plt.title('K-means Clustering with scikit-learn')
plt.show()
```

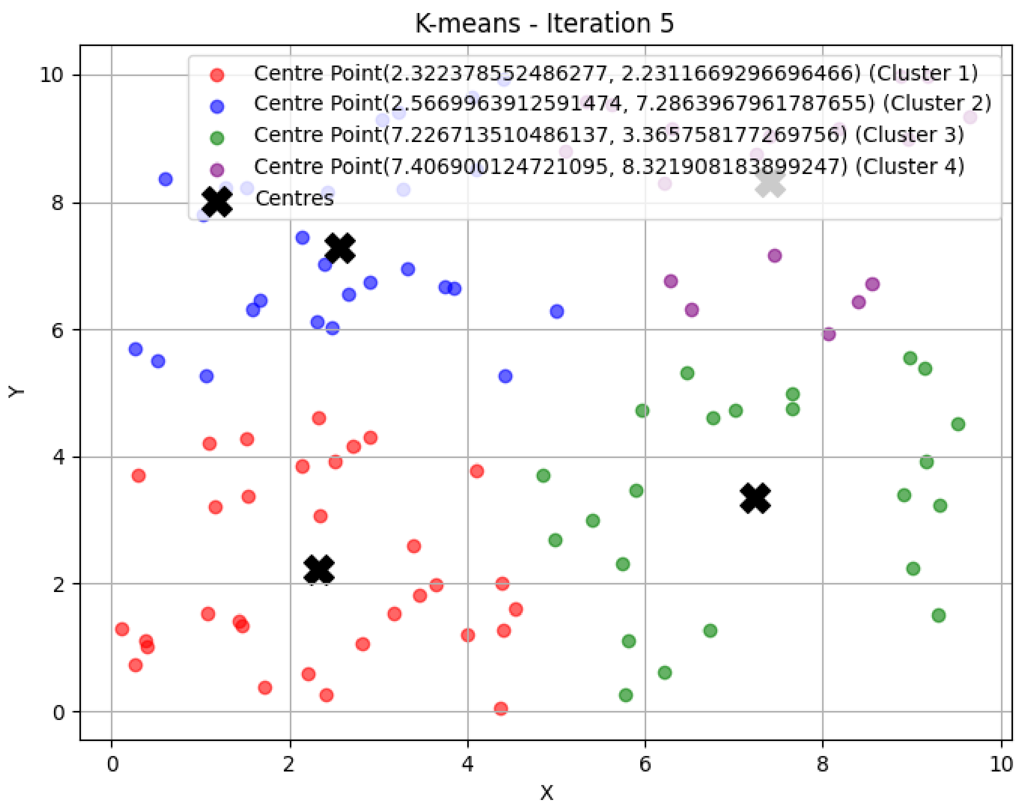

Figure 1: KMeans à la main

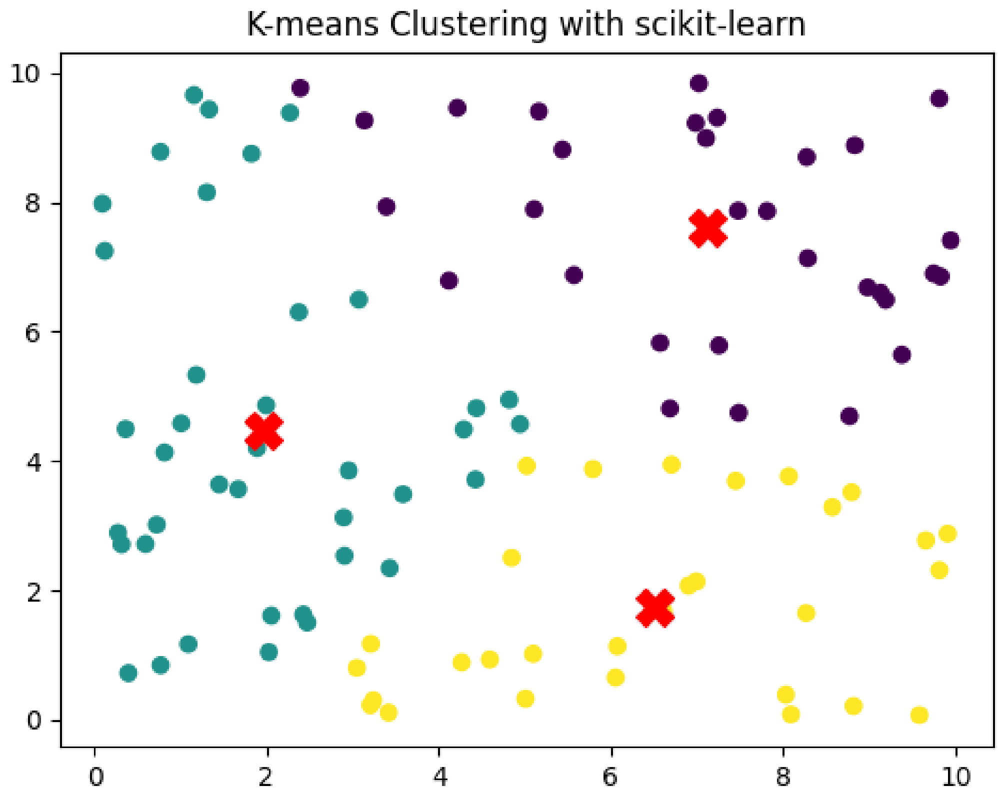

Figure 2: KMeans de sklearn

2 Analyse

En utilisant des codes tests obtenues sur le site de sklean avec de plus en plus de points et changement des centres de départs nous en conclure plusieurs choses.

La rapidité et l'efficacité de l'algorithme changent beaucoup selon les centres de départs:

```
import numpy as np

from sklearn.datasets import load_digits

data, labels = load_digits(return_X_y=True)
(n_samples, n_features), n_digits = data.shape,
    np.unique(labels).size

print(f"# digits: {n_digits}; # samples: {n_samples}; #
```

```
features {n_features}")

from time import time

from sklearn import metrics
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler


def bench_k_means(kmeans, name, data, labels):
    """Benchmark to evaluate the KMeans initialization methods.

    Parameters
    ----------
    kmeans : KMeans instance
        A :class:`~sklearn.cluster.KMeans` instance with the initialization
        already set.
    name : str
        Name given to the strategy. It will be used to show the results in a
        table.
    data : ndarray of shape (n_samples, n_features)
        The data to cluster.
    labels : ndarray of shape (n_samples,)
        The labels used to compute the clustering metrics which requires some
        supervision.
    """
    t0 = time()
    estimator = make_pipeline(StandardScaler(), kmeans).fit(data)
    fit_time = time() - t0
    results = [name, fit_time, estimator[-1].inertia_]
```

```python

    # Define the metrics which require only the true labels
       and estimator
    # labels
    clustering_metrics = [
        metrics.homogeneity_score,
        metrics.completeness_score,
        metrics.v_measure_score,
        metrics.adjusted_rand_score,
        metrics.adjusted_mutual_info_score,
    ]
    results += [m(labels, estimator[-1].labels_) for m in
       clustering_metrics]

    # The silhouette score requires the full dataset
    results += [
        metrics.silhouette_score(
            data,
            estimator[-1].labels_,
            metric="euclidean",
            sample_size=300,
        )
    ]

    # Show the results
    formatter_result = (
        "{:9s}\t{:.3f}s\t{:.0f}\t{:.3f}\t{:.3f}\
        t{:.3f}\t{:.3f}\t{:.3f}\t{:.3f}"
    )
    print(formatter_result.format(*results))

from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

```

```python
print(82 * "_")
print("init\t\ttime\tinertia\thomo
\tcompl\tv-meas\tARI\tAMI\tsilhouette")

kmeans = KMeans(init="k-means++", n_clusters=n_digits,
    n_init=4, random_state=0)
bench_k_means(kmeans=kmeans, name="k-means++", data=data,
    labels=labels)

kmeans = KMeans(init="random", n_clusters=n_digits, n_init=4,
    random_state=0)
bench_k_means(kmeans=kmeans, name="random", data=data,
    labels=labels)

pca = PCA(n_components=n_digits).fit(data)
kmeans = KMeans(init=pca.components_, n_clusters=n_digits,
    n_init=1)
bench_k_means(kmeans=kmeans, name="PCA-based", data=data,
    labels=labels)

print(82 * "_")
```

Ici on teste 3 méthodes pour initialiser les centres : la méthode de kmeans++, une version aléatoire et la version PCA, le code ci-dessus nous donne la sorties suivantes :

```
PS C:\Users\willi\OneDrive\Documents\INFO0503\TP> & C:/Users/willi/AppData/Local/Programs/Python/Python313/python.exe c:/Users/willi/OneDrive/Documents/INFO0503/TP/testKmeans.py
# digits: 10; # samples: 1797; # features 64
__________________________________________________________________________________
init            time    inertia homo    compl   v-meas  ARI     AMI     silhouette
k-means++       0.108s  69545   0.598   0.645   0.621   0.469   0.617   0.146
random          0.030s  69735   0.681   0.723   0.701   0.574   0.698   0.177
PCA-based       0.010s  69513   0.600   0.647   0.622   0.468   0.618   0.146
__________________________________________________________________________________
```

Figure 3: Sortie de code

on remarque que le temps d'exécution varie plus ou moins selon la méthode utilisé et donc incite plus ou moins à changer selon les besoins.

```python
import matplotlib.pyplot as plt

reduced_data = PCA(n_components=2).fit_transform(data)
```

```
kmeans = KMeans(init="k-means++", n_clusters=n_digits,
    n_init=4)
kmeans.fit(reduced_data)

# Step size of the mesh. Decrease to increase the quality of
    the VQ.
h = 0.02  # point in the mesh [x_min, x_max]x[y_min, y_max].

# Plot the decision boundary. For that, we will assign a color
    to each
x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:,
    0].max() + 1
y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:,
    1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
    np.arange(y_min, y_max, h))

# Obtain labels for each point in mesh. Use last trained model.
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1)
plt.clf()
plt.imshow(
    Z,
    interpolation="nearest",
    extent=(xx.min(), xx.max(), yy.min(), yy.max()),
    cmap=plt.cm.Paired,
    aspect="auto",
    origin="lower",
)

plt.plot(reduced_data[:, 0], reduced_data[:, 1], "k.",
```

```
    markersize=2)
# Plot the centroids as a white X
centroids = kmeans.cluster_centers_
plt.scatter(
    centroids[:, 0],
    centroids[:, 1],
    marker="x",
    s=169,
    linewidths=3,
    color="w",
    zorder=10,
)
plt.title(
    "K-means clustering on the digits dataset (PCA-reduced
       data)\n"
    "Centroids are marked with white cross"
)
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.show()
```

En ajoutant cette partie de code on peut donc obtenir l'affichage suivant :

nous montrant les différents clusters de données (chaque couleur) et leur centre (croix blanche).

On remarque aussi une tendance des clusters à ne pas se mélanger, chacun se retrouve de son côté, on le voit sur la figure en dessous mais aussi dans différent test que l'on peut retrouver sur le site de sklearn et leur différent tests.

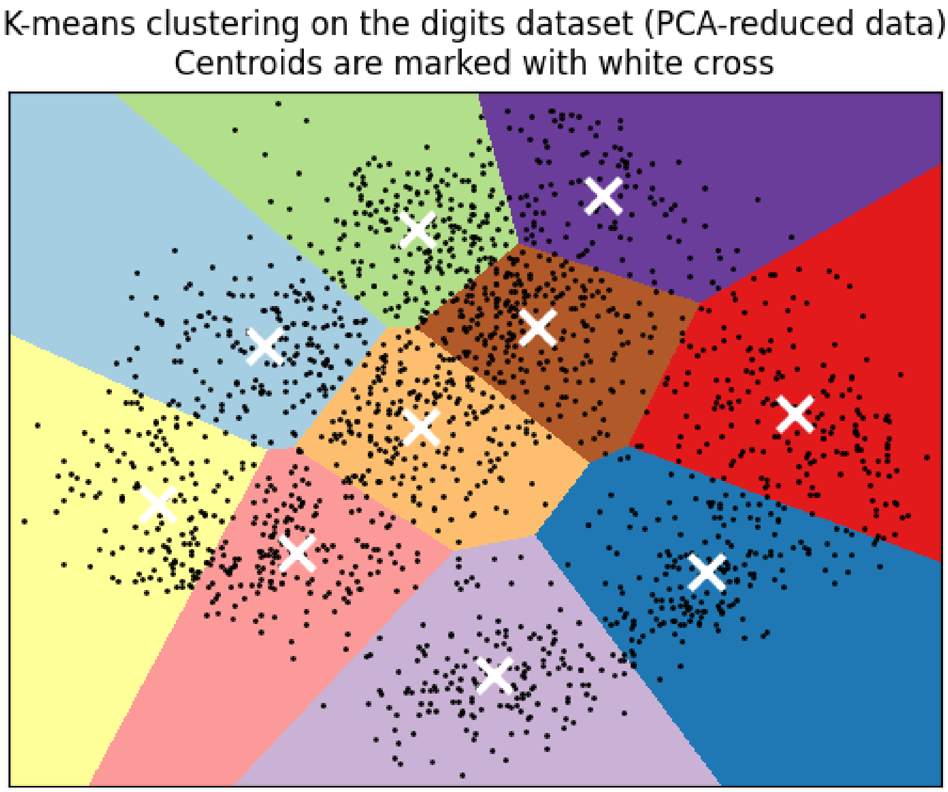

Figure 4: Affichage du code