Лабораторна робота 4 - Навчання без учителя.

У цій лабораторній роботі Ви познайомитеся з основними задачами, які розв'язують алгоритми навчання без учителя.

Завдання 1

Реалізуйте алгоритм кластеризації k-середніх, доповнивши методи, позначені `#TODO` у класі KMeans.

Метод `distance` повинен повертати матрицю $D_{m \times n}$, де m, n - кількість рядків у p_1, p_2 відповідно. $D_{i,j}$ - евклідова відстань між i-м рядком p_1 і j-м рядком p_2. *Порада*: скористайтеся векторизацією, оскільки реалізація через цикли буде значно повільнішою.

Метод `fit` повинен виконувати пошук центроїдів кластерів. Знайдені центроїди мають бути збережені в `self.cluster_centers_` .

Метод `predict` виконує кластеризацію, передбачаючи для кожного елемента `x` індекс відповідного йому кластера.

Виберіть будь-яке зображення (тільки не дуже велике, 64x64 підійде ідеально). За допомогою алгоритму k-середніх підберіть оптимальну кількість кластерів для кластеризації пікселів зображення, максимізуючи silhouette-score. Візуалізуйте кластеризацію з найкращим k.

Код класу OurKMeans:

```python
import numpy as np
import matplotlib.pyplot as plt
from skimage.io import imread
from skimage.transform import resize
from sklearn.metrics import silhouette_score
from math import dist
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import warnings
warnings.filterwarnings("ignore")
import random
import numpy as np
from datetime import datetime
from datetime import timedelta
class OurKMeans:
    def __init__(self, n_clusters):
        self.n_clusters = n_clusters
    @staticmethod
    def distance(p1, p2):
        # D=np.zeros((len(p1),len(p2)))
        # for i in range(len(p1)):
            # for j in range(len(p2)):
                # D[i,j]=sum((y-x)**2 for x, y in zip(p1[i], p2[j])) ** 0.5
        D = np.linalg.norm(p1[:, np.newaxis] - p2, axis=2)
        return D
```

```
    def fit(self, x):
        count_of_samples = len(x)
        count_of_features = len(x[0])
        self.cluster_centers_ = np.empty((self.n_clusters, count_of_features))
        self.cluster_centers_[0] = x[np.random.choice(count_of_samples)]
        for i in range(1, self.n_clusters):
            distances = np.min(self.distance(x, self.cluster_centers_[:i]), axis=1)
            next_centroid = np.argmax(distances)
            self.cluster_centers_[i] = x[next_centroid]
        while True:
            D = np.argmin(self.distance(x, self.cluster_centers_),axis=1)
            new_cluster_centers = []
            for i in range(self.n_clusters):
                new_cluster_center = x[D == i]
                if(len(new_cluster_center) > 0):
                    new_cluster_centers.append(new_cluster_center.mean(axis=0))
                else:
                    new_cluster_centers.append(x[np.random.choice(x.shape[0])])
            if np.allclose(new_cluster_centers, self.cluster_centers_):
                break
            self.cluster_centers_ = new_cluster_centers
        return self
    def predict(self, x):
        D = self.distance(x, self.cluster_centers_)
        return D.argmin(axis=1)
```

Опис проведених досліджень

KMeans, кластеризація методом К-середніх – популярний метод кластеризації, тобто впорядкування множини об'єктів у порівняно однорідні групи.

Реалізація алгоритму KMeans складається з таких основних частин:

1. Обчислення Евклідової відстані між двома точками за формулою:

$$\text{distance}(p_1, p_2) = \sqrt{\sum_i (p_{1,i} - p_{2,i})^2}$$

2. Ініціалізація центрів кластерів. Спочатку використовувалось таке рішення:

```
random_indices = np.random.choice(x.shape[0], self.n_clusters, replace=False)
self.cluster_centers_ = x[random_indices]
```

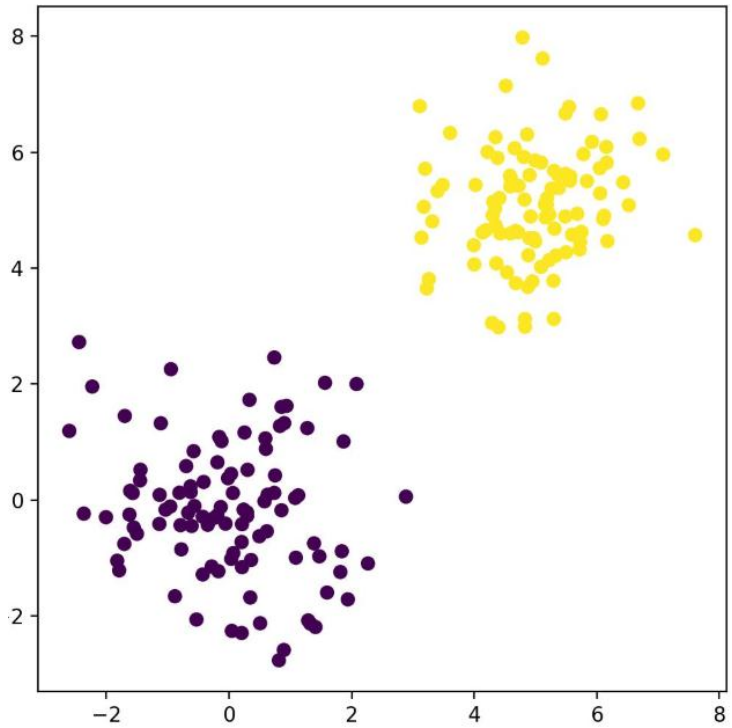
Воно коректно працює лише для набору випадкових даних, а при обробці зображення виникає помилка «Error: Number of labels is 1. Valid values are 2 to n_samples – 1». Тому ми використовували інший алгоритм, K-means++. Він є більш розумним способом ініціалізації центроїдів. Замість того, щоб обирати їх випадковим чином, він випадково визначає лише перший, а наступні – на основі максимальної відстані від вже визначених. Для зменшення ймовірності виникнення помилки у коді також було додано обробку порожніх кластерів.

3. Призначення кожної точки набору даних x найближчому центру кластера, що виконується шляхом обчислення мінімальної відстані для кожної точки.

4. Оновлення центрів: після присвоєння точок кластерам ми обчислюємо нові центри кластерів як середні значення точок, призначених кожному кластеру.

5. Перевірка збіжності: обчислення виконуються, доки центри не зміняться.

При перевірці роботи методів класу на випадкових даних при $K = 2$ було отримано такий графічний результат:



Для тестування класу використовувалось зображення розміру 64 на 64 у форматах bmp та jpg. У процесі виконання даної лабораторної роботи ми порівнювали silhouette-score, отримане нашим рішенням, із результатами роботи аналогічного інструмента KMeans із бібліотеки Sklearn. Також було реалізовано дві версії метода обчислення відстані: на основі циклів із оптимізацією обчислення квадратного кореня шляхом піднесення підкореневого виразу у степінь 0.5 замість використання команди sqrt, та команди np.linalg.norm.

Результати роботи програми із обчисленням відстані за допомогою циклів:

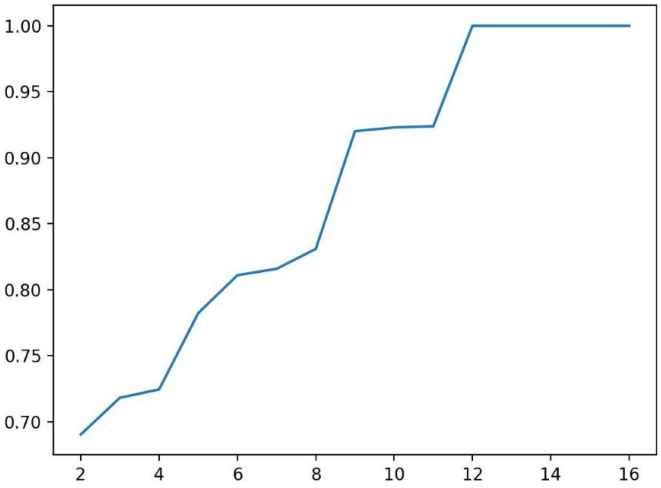
```
K:  2 ; KMeans silhouette score:  0.69035; Sklearn KMeans silhouette score:  0.69035
K:  3 ; KMeans silhouette score:  0.74551; Sklearn KMeans silhouette score:  0.74742
K:  4 ; KMeans silhouette score:  0.80968; Sklearn KMeans silhouette score:  0.80968
K:  5 ; KMeans silhouette score:  0.78233; Sklearn KMeans silhouette score:  0.83032
K:  6 ; KMeans silhouette score:  0.79787; Sklearn KMeans silhouette score:  0.88518
K:  7 ; KMeans silhouette score:  0.83290; Sklearn KMeans silhouette score:  0.93127
K:  8 ; KMeans silhouette score:  0.83095; Sklearn KMeans silhouette score:  0.95227
K:  9 ; KMeans silhouette score:  0.92012; Sklearn KMeans silhouette score:  0.99127
K:  10 ; KMeans silhouette score:  0.92298; Sklearn KMeans silhouette score:  0.99632
K:  11 ; KMeans silhouette score:  0.92380; Sklearn KMeans silhouette score:  0.99918
K:  12 ; KMeans silhouette score:  1.00000; Sklearn KMeans silhouette score:  1.00000
K:  13 ; KMeans silhouette score:  1.00000; Sklearn KMeans silhouette score:  1.00000
K:  14 ; KMeans silhouette score:  1.00000; Sklearn KMeans silhouette score:  1.00000
K:  15 ; KMeans silhouette score:  1.00000; Sklearn KMeans silhouette score:  1.00000
K:  16 ; KMeans silhouette score:  1.00000; Sklearn KMeans silhouette score:  1.00000
Time:  0 : 0 : 31
```

Результати роботи програми із обчисленням відстані без циклів:

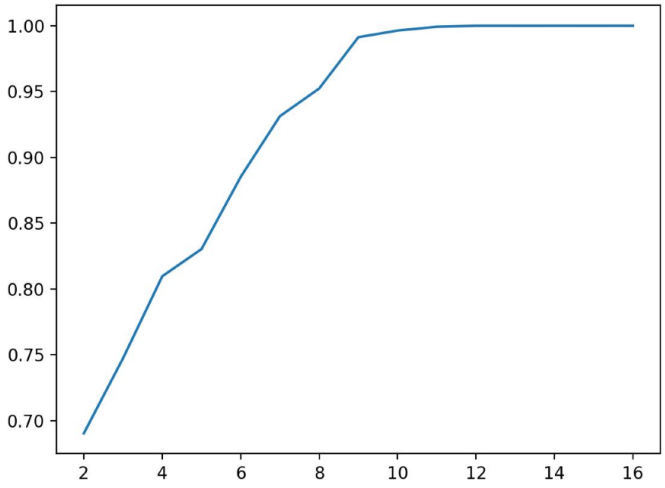
```
K:  2 ; KMeans silhouette score:  0.69035; Sklearn KMeans silhouette score:  0.69035
K:  3 ; KMeans silhouette score:  0.71822; Sklearn KMeans silhouette score:  0.74742
K:  4 ; KMeans silhouette score:  0.72450; Sklearn KMeans silhouette score:  0.80968
K:  5 ; KMeans silhouette score:  0.78233; Sklearn KMeans silhouette score:  0.83032
K:  6 ; KMeans silhouette score:  0.81101; Sklearn KMeans silhouette score:  0.88518
K:  7 ; KMeans silhouette score:  0.81587; Sklearn KMeans silhouette score:  0.93127
K:  8 ; KMeans silhouette score:  0.83095; Sklearn KMeans silhouette score:  0.95227
K:  9 ; KMeans silhouette score:  0.92012; Sklearn KMeans silhouette score:  0.99127
K:  10 ; KMeans silhouette score:  0.92298; Sklearn KMeans silhouette score:  0.99632
K:  11 ; KMeans silhouette score:  0.92380; Sklearn KMeans silhouette score:  0.99918
K:  12 ; KMeans silhouette score:  1.00000; Sklearn KMeans silhouette score:  1.00000
K:  13 ; KMeans silhouette score:  1.00000; Sklearn KMeans silhouette score:  1.00000
K:  14 ; KMeans silhouette score:  1.00000; Sklearn KMeans silhouette score:  1.00000
K:  15 ; KMeans silhouette score:  1.00000; Sklearn KMeans silhouette score:  1.00000
K:  16 ; KMeans silhouette score:  1.00000; Sklearn KMeans silhouette score:  1.00000
Time:  0 : 0 : 11
```

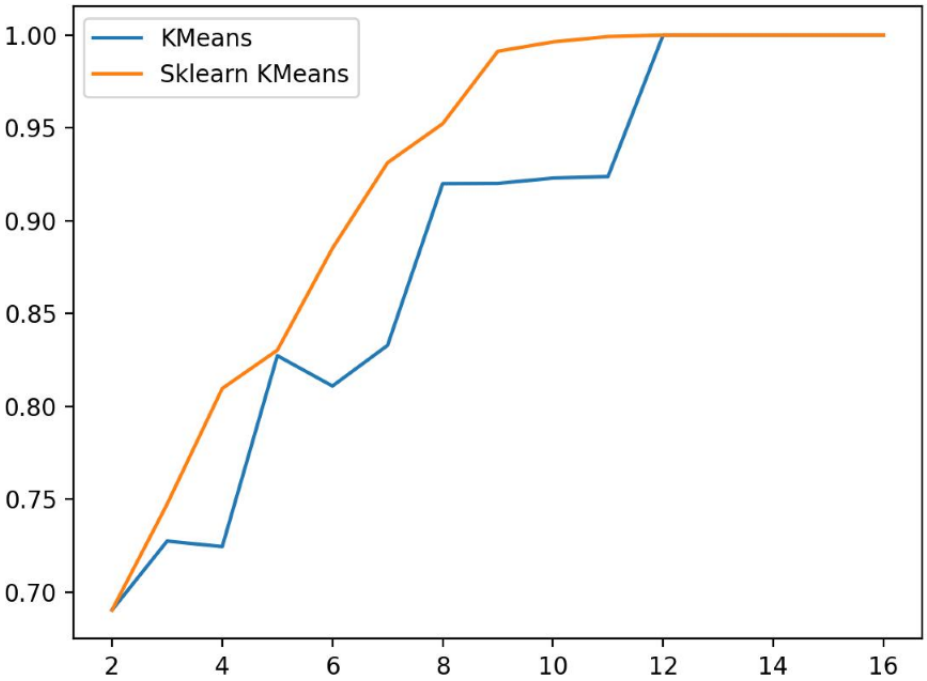
Як бачимо, версія із циклами працює майже в три рази повільніше, ніж відповідна функція бібліотеки numpy. Різні способи обчислення відстані дають різні результати при деяких значеннях *K*, а функція KMeans бібліотеки Sklearn показує кращий silhouette-score при всіх K окрім 2, що помітно на графіках:

KMeans



Sklearn KMeans





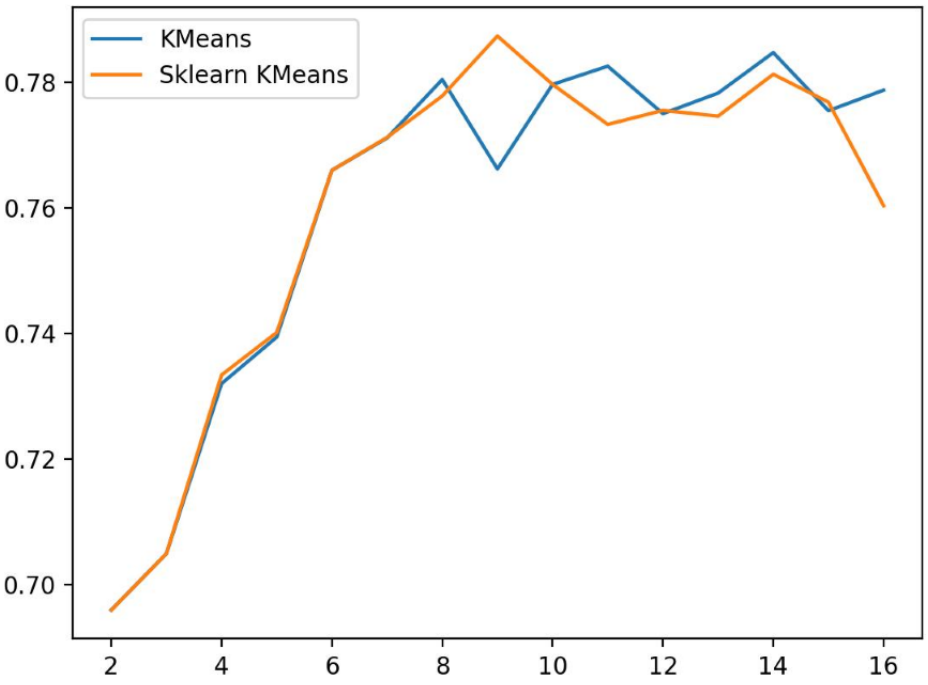
Результати експериментів із тим самим зображенням, але у форматі jpg:

```
K:  2 ; KMeans silhouette score:  0.69598; Sklearn KMeans silhouette score:  0.69598
K:  3 ; KMeans silhouette score:  0.67805; Sklearn KMeans silhouette score:  0.70497
K:  4 ; KMeans silhouette score:  0.73206; Sklearn KMeans silhouette score:  0.73344
K:  5 ; KMeans silhouette score:  0.73947; Sklearn KMeans silhouette score:  0.74021
K:  6 ; KMeans silhouette score:  0.75026; Sklearn KMeans silhouette score:  0.76605
K:  7 ; KMeans silhouette score:  0.77120; Sklearn KMeans silhouette score:  0.77128
K:  8 ; KMeans silhouette score:  0.76030; Sklearn KMeans silhouette score:  0.77789
K:  9 ; KMeans silhouette score:  0.76626; Sklearn KMeans silhouette score:  0.78744
K:  10 ; KMeans silhouette score:  0.76716; Sklearn KMeans silhouette score:  0.77973
K:  11 ; KMeans silhouette score:  0.77548; Sklearn KMeans silhouette score:  0.77336
K:  12 ; KMeans silhouette score:  0.77507; Sklearn KMeans silhouette score:  0.77559
K:  13 ; KMeans silhouette score:  0.77835; Sklearn KMeans silhouette score:  0.77467
K:  14 ; KMeans silhouette score:  0.78533; Sklearn KMeans silhouette score:  0.78134
K:  15 ; KMeans silhouette score:  0.78311; Sklearn KMeans silhouette score:  0.77690
K:  16 ; KMeans silhouette score:  0.77890; Sklearn KMeans silhouette score:  0.76040
Time:  0 : 0 : 39
```

```
K:  2 ; KMeans silhouette score:  0.69598; Sklearn KMeans silhouette score:  0.69598
K:  3 ; KMeans silhouette score:  0.67805; Sklearn KMeans silhouette score:  0.70497
K:  4 ; KMeans silhouette score:  0.73344; Sklearn KMeans silhouette score:  0.73344
K:  5 ; KMeans silhouette score:  0.73947; Sklearn KMeans silhouette score:  0.74021
K:  6 ; KMeans silhouette score:  0.76605; Sklearn KMeans silhouette score:  0.76605
K:  7 ; KMeans silhouette score:  0.77126; Sklearn KMeans silhouette score:  0.77128
K:  8 ; KMeans silhouette score:  0.76030; Sklearn KMeans silhouette score:  0.77789
K:  9 ; KMeans silhouette score:  0.76851; Sklearn KMeans silhouette score:  0.78744
K:  10 ; KMeans silhouette score:  0.77973; Sklearn KMeans silhouette score:  0.77973
K:  11 ; KMeans silhouette score:  0.77548; Sklearn KMeans silhouette score:  0.77336
K:  12 ; KMeans silhouette score:  0.77507; Sklearn KMeans silhouette score:  0.77559
K:  13 ; KMeans silhouette score:  0.77835; Sklearn KMeans silhouette score:  0.77467
K:  14 ; KMeans silhouette score:  0.77733; Sklearn KMeans silhouette score:  0.78134
K:  15 ; KMeans silhouette score:  0.78217; Sklearn KMeans silhouette score:  0.77690
K:  16 ; KMeans silhouette score:  0.77890; Sklearn KMeans silhouette score:  0.76040
Time:  0 : 0 : 11
```

Як бачимо, загальний рівень silhouette-score для jpg нижчий за bmp. Також слід відмітити різні значення цього коефіцієнта у нашій реалізації при обчисленні відстані з циклами та без них. Співвідношення швидкодії приблизно дорівнює bmp.

На цьому графіку можна помітити, що у деяких випадках наша реалізація KMeans дає кращий silhouette-score:



Завдання 2

Реалізуйте алгоритм аналізу головних компонент, доповнивши методи, позначені `#TODO` у класі PCA, і застосуйте його на датасеті MNIST.

Метод `fit` повинен виконувати пошук `self.n_component` головних компонент:

$$X = [x_{i,j}]_{m \times n}$$

$$X_c = \left[x_{i,j} - \frac{1}{m} \sum_{k=1}^{m} x_{k,j} \right]$$

$$C = \frac{X_c^T X_c}{m - 1}$$

Головними компонентами будуть власні вектори матриці C, які відповідають `self.n_component` найбільшим власним числам. (Скористайтеся np.linalg.eig).

Знайдені компоненти мають бути збережені в `self.components_` .

Метод `transform` виконує проекцію даних:

$$\left[y_{i,j} - \frac{1}{m} \sum_{k=1}^{m} x_{k,j} \right] \times V$$

де y - елементи матриці даних, для яких виконується проекція, V - матриця, сформована з головних компонент. Зверніть увагу, що для нормалізації використовуються середні за тренувальною вибіркою.

Код класу OurPCA:

```
 98 class OurPCA:
 99     def __init__(self, n_components):
100         self.n_components = n_components
101     def fit(self, X):
102         self.mean_ = X.mean(axis=0)
103         X_centered = X - self.mean_
104         covariance_matrix = np.cov(X_centered.T)
105         eigenvalue, eigenvectors = np.linalg.eigh(covariance_matrix)
106         sorted_indices = np.argsort(eigenvalue)[::-1]
107         self.components_ = eigenvectors[:, sorted_indices[:self.n_components]]
108         return self
109     def transform(self, X):
110         X_centered = X - self.mean_
111         X_proj = X_centered.dot(self.components_)
112         return X_proj
```

Опис проведених досліджень

PCA (principal component analysis), метод головних компонент – один із основних способів зменшення розмірності даних із втратою найменшої кількості інформації. Його реалізація складається з таких основних етапів:

1. центрування даних (віднімання середнього значення кожної ознаки);

2. обчислення коваріаційної матриці;

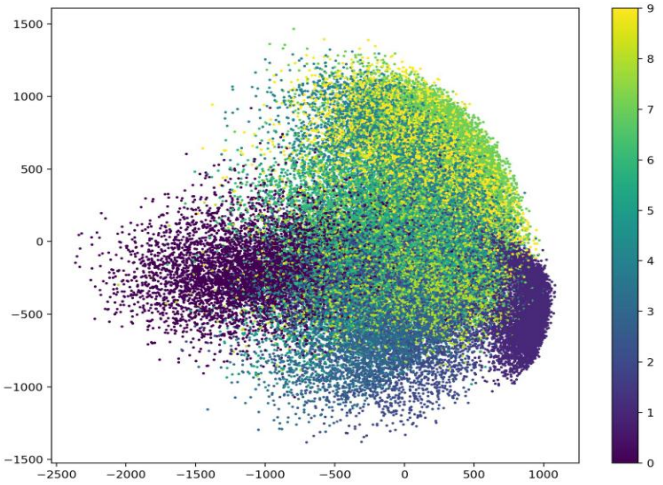
3. розкладання коваріаційної матриці на власні значення (eigenvalues);

4. сортування векторів власних значень (eigenvectors) за власними значеннями (eigenvalues) в порядку спадання;

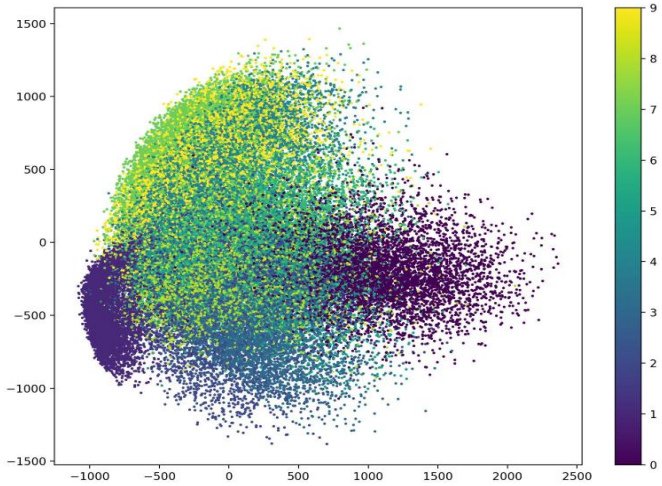
5. Проекція даних на основні компоненти (метод *transform*).

У рамках даної частини лабораторної роботи ми порівняли нашу реалізацію та подібний інструмент з бібліотеки Sklearn, проаналізувавши датасет MNIST:

PCA



Sklearn PCA



Як бачимо, графічний результат роботи нашої версії PCA є дзеркальним відображенням Sklearn PCA.