Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра комп’ютерного моделювання процесів і систем

ЗВІТ

з лабораторної роботи №10

“Зниження розмірності”

з курсу

«Алгоритми та моделі збору, аналізу та візуалізації даних»

Виконав: студент групи ІКМ-М222к  Черкас Ю.В.

Перевірила: аспірантка  Рикова В.О.

Харків 2023р

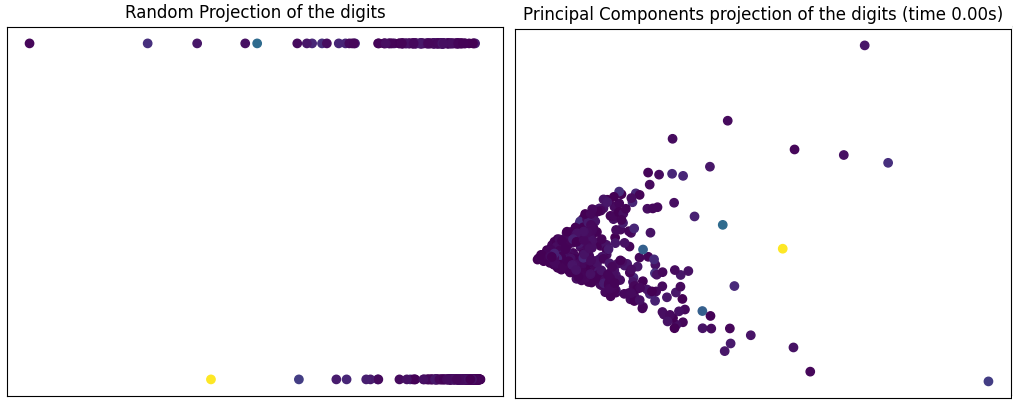
**Варіант №15**

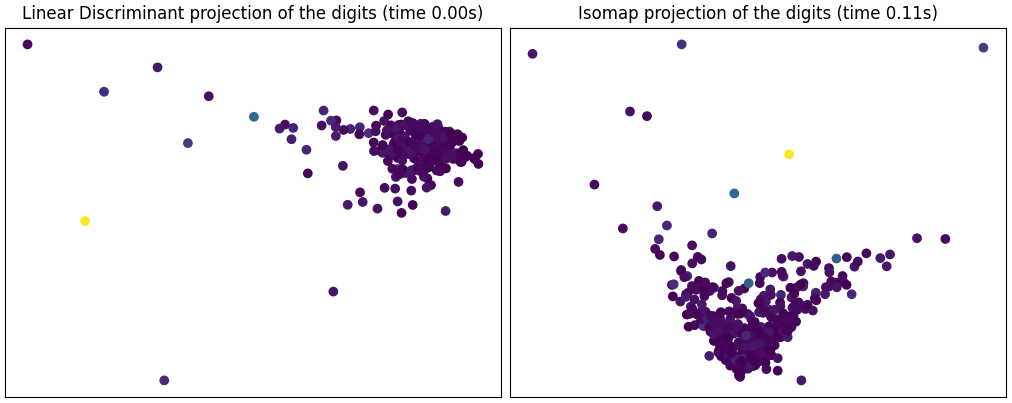
Для відповідного датасету згідно з варіантом виконати пониження розмірності даних до просторів з розмірностями два та три (2D та 3D). Для пониження розмірності використовуйте всі доступні методи бібліотеки scikit-learn для зниження розмірності. Порівняйте результати та визначте яким методом було досягнуто найкращий результат.

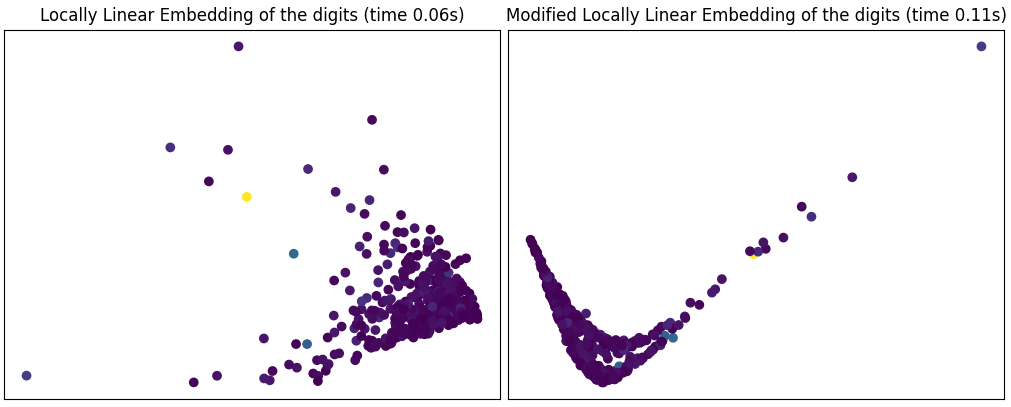
**Виконання**

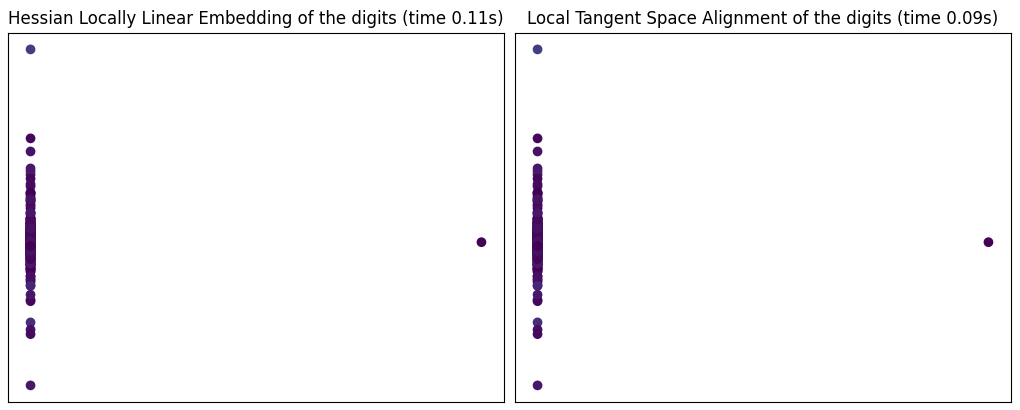
Лістинг програми наведений в Додатку А.

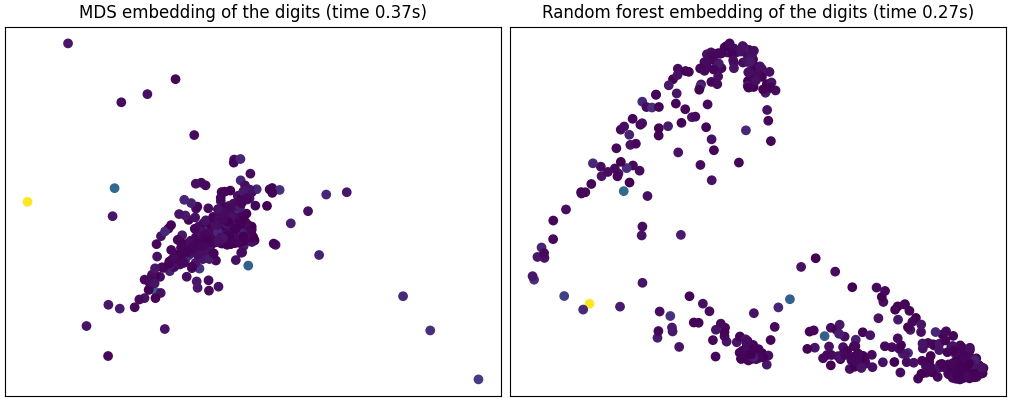
**Графіки пониження розмірності 2D**

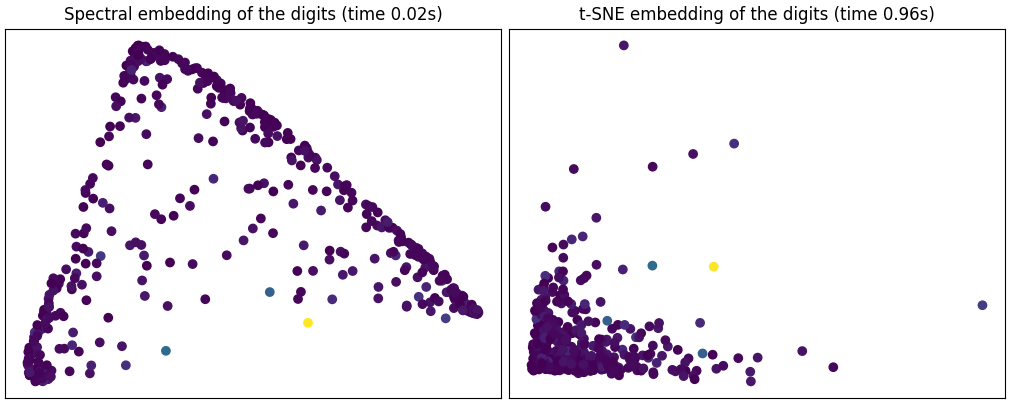




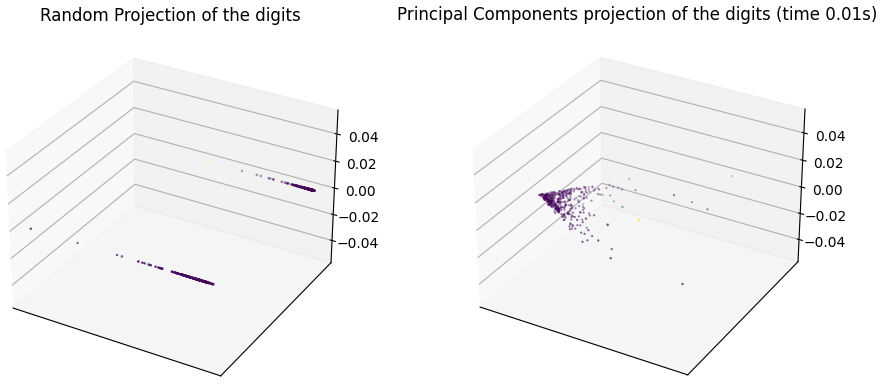


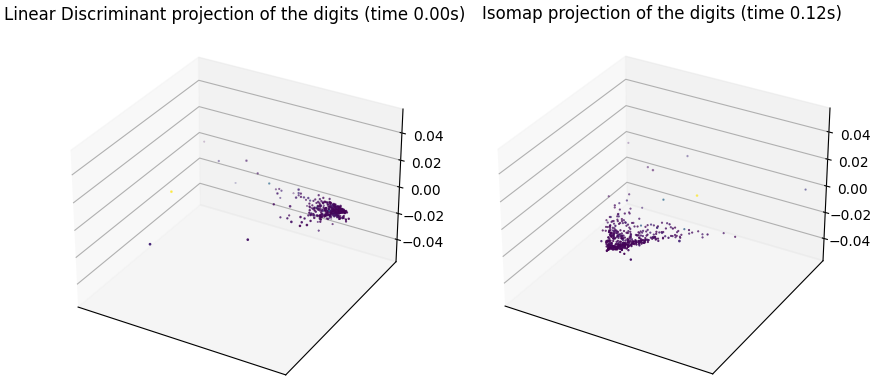


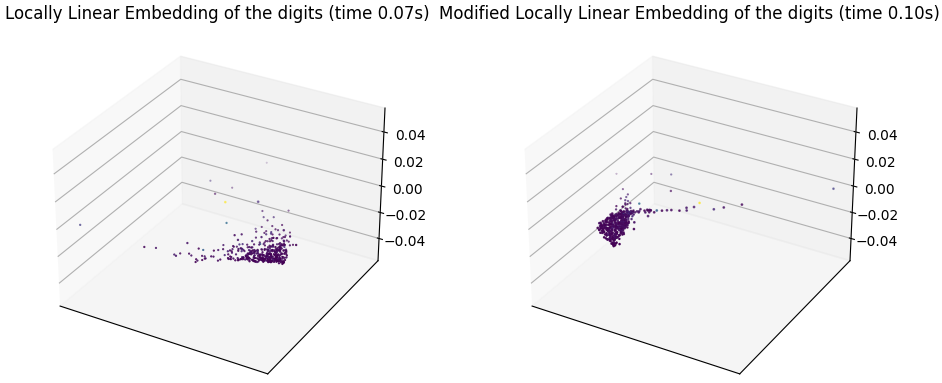


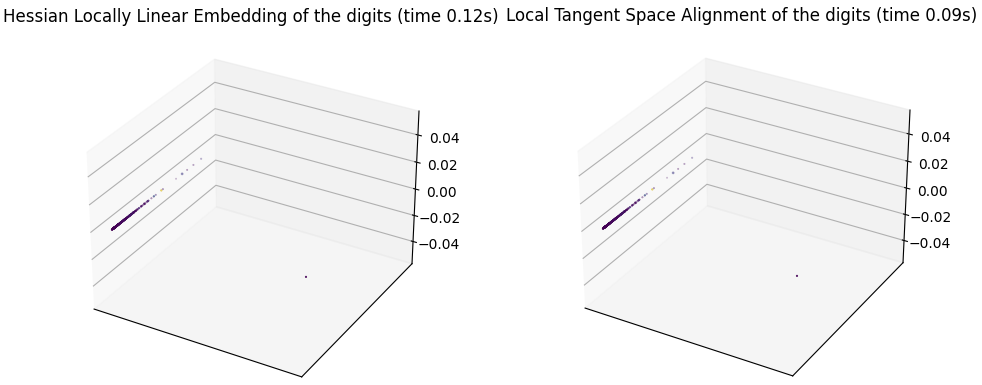


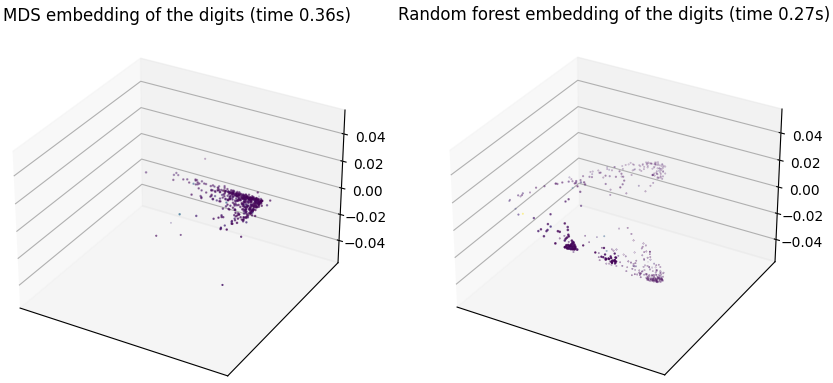
**Графіки пониження розмірності 3D**

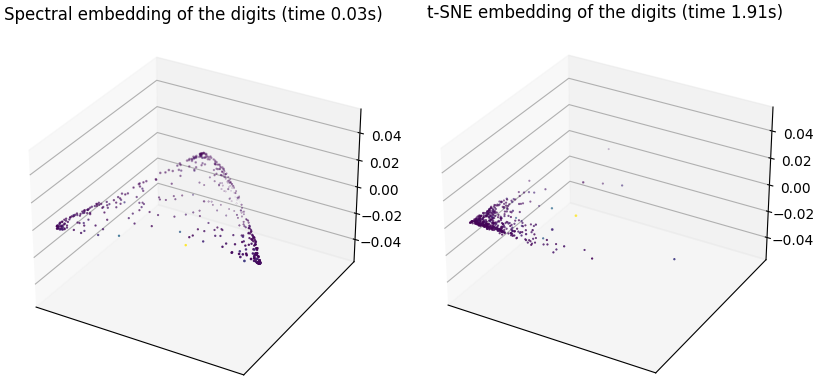












**Висновок:** на даній лабораторній роботі ми дослідили різні методи пониження розмірності даних. Використання даних методів дозволяє більш ефективніше аналізувати дані високої розмірності. Побачили, що візуалізація одних і тих же даних кардинально відрізняється від методу до методу. Проте результати аналізу залежать не тільки від методів пониження розмірності, а і від характеру самих даних. Для даних аналізованих в даній роботі ('Wholesale customers data.csv' – дані оптових продаж), відсутня явно виражена категоризація. Проте даний факт не применшує можливість аналізу загальної структури вибірки даних в цілому.

**Додаток А**

Лістинг програми

# Authors: Fabian Pedregosa <fabian.pedregosa@inria.fr>

#          Olivier Grisel <olivier.grisel@ensta.org>

#          Mathieu Blondel <mathieu@mblondel.org>

#          Gael Varoquaux

# License: BSD 3 clause (C) INRIA 2011

print(\_\_doc\_\_)

from time import time

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from matplotlib import offsetbox

from sklearn import (manifold, datasets, decomposition, ensemble, discriminant\_analysis, random\_projection)

digits = pd.read\_csv('Wholesale customers data.csv')

X = digits.iloc[:,:-1].values.astype(np.float)

y = digits.iloc[:,-1].values

n\_samples, n\_features = X.shape

n\_neighbors = 30

#---------------------------------------------------------------

# Scale and visualize the embedding vectors

def plot\_embedding(X, title=None):

    x\_min, x\_max = np.min(X, 0), np.max(X, 0)

    X = (X - x\_min) / (x\_max - x\_min)

    plt.figure()

    ax = plt.subplot(111)

    plt.scatter(X[:,0], X[:,1], c = y)

    plt.xticks([]), plt.yticks([])

    if title is not None:

        plt.title(title)

#---------------------------------------------------------------

# Random 2D projection using a random unitary matrix

print("Computing random projection")

rp = random\_projection.SparseRandomProjection(n\_components=2, random\_state=42)

X\_projected = rp.fit\_transform(X)

plot\_embedding(X\_projected, "Random Projection of the digits")

#---------------------------------------------------------------

# Projection on to the first 2 principal components

print("Computing PCA projection")

t0 = time()

X\_pca = decomposition.TruncatedSVD(n\_components=2).fit\_transform(X)

plot\_embedding(X\_pca,

               "Principal Components projection of the digits (time %.2fs)" %

               (time() - t0))

#----------------------------------------------------------------------

# Projection on to the first 2 linear discriminant components

print("Computing Linear Discriminant Analysis projection")

X2 = X.copy()

X2.flat[::X.shape[1] + 1] += 0.01  # Make X invertible

t0 = time()

X\_lda = discriminant\_analysis.LinearDiscriminantAnalysis(n\_components=2).fit\_transform(X2, y)

plot\_embedding(X\_lda,

               "Linear Discriminant projection of the digits (time %.2fs)" %

               (time() - t0))

#---------------------------------------------------------------

# Isomap projection of the digits dataset

print("Computing Isomap embedding")

t0 = time()

X\_iso = manifold.Isomap(n\_neighbors = n\_neighbors, n\_components=2).fit\_transform(X)

print("Done.")

plot\_embedding(X\_iso,

               "Isomap projection of the digits (time %.2fs)" %

               (time() - t0))

#---------------------------------------------------------------

# Locally linear embedding of the digits dataset

print("Computing LLE embedding")

clf = manifold.LocallyLinearEmbedding(n\_neighbors = n\_neighbors, n\_components=2,

                                      method='standard')

t0 = time()

X\_lle = clf.fit\_transform(X)

print("Done. Reconstruction error: %g" % clf.reconstruction\_error\_)

plot\_embedding(X\_lle,

               "Locally Linear Embedding of the digits (time %.2fs)" %

               (time() - t0))

#---------------------------------------------------------------

# Modified Locally linear embedding of the digits dataset

print("Computing modified LLE embedding")

clf = manifold.LocallyLinearEmbedding(n\_neighbors = n\_neighbors, n\_components=2,

                                      method='modified')

t0 = time()

X\_mlle = clf.fit\_transform(X)

print("Done. Reconstruction error: %g" % clf.reconstruction\_error\_)

plot\_embedding(X\_mlle,

               "Modified Locally Linear Embedding of the digits (time %.2fs)" %

               (time() - t0))

#---------------------------------------------------------------

# HLLE embedding of the digits dataset

print("Computing Hessian LLE embedding")

clf = manifold.LocallyLinearEmbedding(n\_neighbors = n\_neighbors, n\_components=2,

                                      method='hessian', eigen\_solver = 'dense')

t0 = time()

X\_hlle = clf.fit\_transform(X)

print("Done. Reconstruction error: %g" % clf.reconstruction\_error\_)

plot\_embedding(X\_hlle,

               "Hessian Locally Linear Embedding of the digits (time %.2fs)" %

               (time() - t0))

#---------------------------------------------------------------

# LTSA embedding of the digits dataset

print("Computing LTSA embedding")

clf = manifold.LocallyLinearEmbedding(n\_neighbors = n\_neighbors, n\_components=2,

                                      method='ltsa', eigen\_solver = 'dense')

t0 = time()

X\_ltsa = clf.fit\_transform(X)

print("Done. Reconstruction error: %g" % clf.reconstruction\_error\_)

plot\_embedding(X\_ltsa,

               "Local Tangent Space Alignment of the digits (time %.2fs)" %

               (time() - t0))

#---------------------------------------------------------------

# MDS  embedding of the digits dataset

print("Computing MDS embedding")

clf = manifold.MDS(n\_components=2, n\_init=1, max\_iter=100)

t0 = time()

X\_mds = clf.fit\_transform(X)

print("Done. Stress: %f" % clf.stress\_)

plot\_embedding(X\_mds,

               "MDS embedding of the digits (time %.2fs)" %

               (time() - t0))

#---------------------------------------------------------------

# Random Trees embedding of the digits dataset

print("Computing Totally Random Trees embedding")

hasher = ensemble.RandomTreesEmbedding(n\_estimators=200, random\_state=0,

                                       max\_depth=5)

t0 = time()

X\_transformed = hasher.fit\_transform(X)

pca = decomposition.TruncatedSVD(n\_components=2)

X\_reduced = pca.fit\_transform(X\_transformed)

plot\_embedding(X\_reduced,

               "Random forest embedding of the digits (time %.2fs)" %

               (time() - t0))

#---------------------------------------------------------------

# Spectral embedding of the digits dataset

print("Computing Spectral embedding")

embedder = manifold.SpectralEmbedding(n\_components=2, random\_state=0,

                                      eigen\_solver="arpack")

t0 = time()

X\_se = embedder.fit\_transform(X)

plot\_embedding(X\_se,

               "Spectral embedding of the digits (time %.2fs)" %

               (time() - t0))

#---------------------------------------------------------------

# t-SNE embedding of the digits dataset

print("Computing t-SNE embedding")

tsne = manifold.TSNE(n\_components=2, init='pca', random\_state=0)

t0 = time()

X\_tsne = tsne.fit\_transform(X)

plot\_embedding(X\_tsne,

               "t-SNE embedding of the digits (time %.2fs)" %

               (time() - t0))

plt.show()