Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

По дисциплине: «Интеллектуальный анализ данных»

Тема: «**Автоэнкодеры**»

Выполнил:

Студент 4 курса

Группы ИИ-23

Глухарев Д.Е.

Проверила:

Андренко К.В.

Брест 2025

Цель: научиться применять автоэнкодеры для осуществления визуализации данных и их анализа

Общее задание

1. Используя выборку по варианту, осуществить проецирование данных на плоскость первых двух и трех главных компонент с использованием нейросетевой модели автоэнкодера (с двумя и тремя нейронами в среднем слое);
2. Выполнить визуализацию полученных главных компонент с использованием средств библиотеки matplotlib, обозначая экземпляры разных классов с использованием разных цветовых маркеров;
3. Реализовать метод t-SNE для визуализации данных (использовать также 2 и 3 компонента), построить соответствующую визуализацию;
4. Применить к данным метод PCA (2 и 3 компонента), реализованный в ЛР №1, сделать выводы;
5. Оформить отчет по выполненной работе, загрузить исходный код и отчет в соответствующий репозиторий на github.

Задание по вариантам

5	Optical recognition of handwritten digits	Последний признак (tra)

Код программы:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot
as plt
from
sklearn.decomposition
import PCA
from sklearn.manifold
import TSNE
from
sklearn.preprocessing
import StandardScaler
from sklearn.datasets
import load_digits
from
mpl_toolkits.mplot3d
import Axes3D
import torch
import torch.nn as nn
import torch.optim as
optim

digits = load_digits()
X = digits.data  # (1797,
64)
y = digits.target  # (1797,)
— от 0 до 9
```

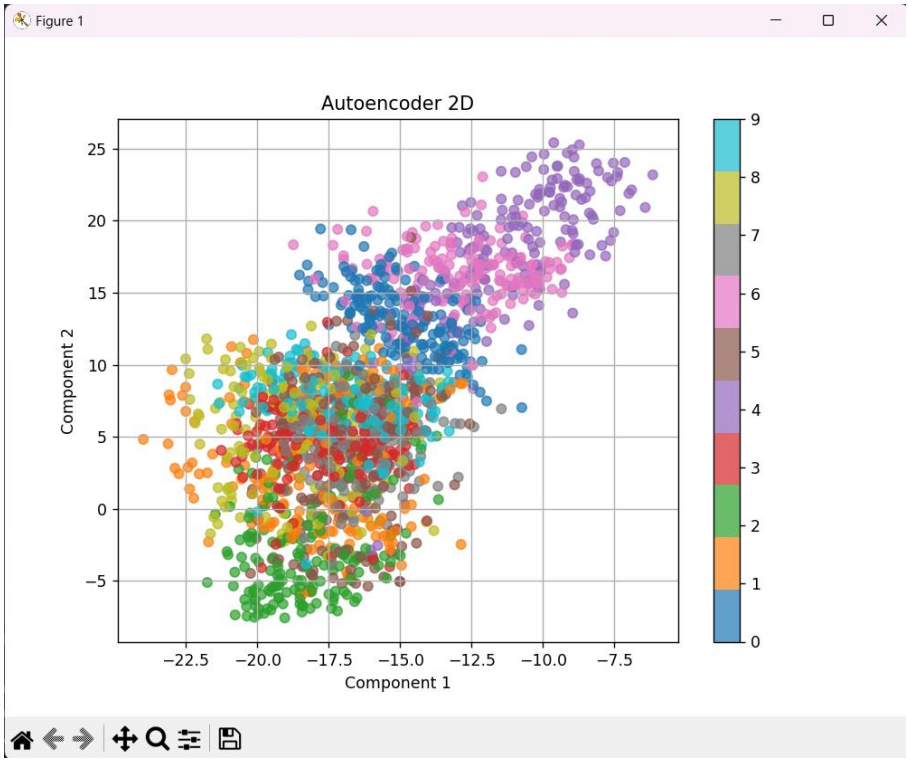
```
def train_autoencoder(X,
n_components,
epochs=200, lr=1e-3):
    X_tensor =
torch.FloatTensor(X)
    input_dim =
X.shape[1]

    class
Autoencoder(nn.Module):
        def __init__(self,
input_dim, hidden_dim):

super(Autoencoder,
self).__init__()
            self.encoder =
nn.Sequential(

nn.Linear(input_dim,
128),
                nn.ReLU(),
                nn.Linear(128,
64),
                nn.ReLU(),
                nn.Linear(64,
hidden_dim)
            )
            self.decoder =
nn.Sequential(

nn.Linear(hidden_dim,
64),
                nn.ReLU(),
                nn.Linear(64,
128),
                nn.ReLU(),
                nn.Linear(128,
input_dim)
            )

        def forward(self, x):
            encoded =
self.encoder(x)
            decoded =
self.decoder(encoded)
            return decoded,
encoded
```

```
    model =
Autoencoder(input_dim,
n_components)
    criterion =
nn.MSELoss()
    optimizer =
optim.Adam(model.para
meters(), lr=lr)

    for epoch in
range(epochs):
        decoded, encoded =
model(X_tensor)
        loss =
criterion(decoded,
X_tensor)

optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if epoch % 50 == 0:
            print(f'Epoch
[{epoch}/{epochs}],
Loss: {loss.item():.4f}')

    with torch.no_grad():
        _, encoded =
model(X_tensor)
    return
encoded.numpy()

X_ae_2d =
train_autoencoder(X, 2)
X_ae_3d =
train_autoencoder(X, 3)

def plot_2d_scatter(X, y,
title):
    plt.figure(figsize=(8,
6))
    scatter = plt.scatter(X[:,
0], X[:, 1], c=y,
cmap='tab10', alpha=0.7)
    plt.colorbar(scatter)
    plt.title(title)
    plt.xlabel('Component
1')
    plt.ylabel('Component
2')
```

```
    plt.grid(True)
    plt.show()

def plot_3d_scatter(X, y,
title):
    fig =
plt.figure(figsize=(10, 7))
    ax =
fig.add_subplot(111,
projection='3d')
    scatter = ax.scatter(X[:,
0], X[:, 1], X[:, 2], c=y,
cmap='tab10', alpha=0.7)
    plt.title(title)

ax.set_xlabel('Component
1')

ax.set_ylabel('Component
2')

ax.set_zlabel('Component
3')
    plt.show()

plot_2d_scatter(X_ae_2d,
y, 'Autoencoder 2D')
plot_3d_scatter(X_ae_3d,
y, 'Autoencoder 3D')

X_scaled =
StandardScaler().fit_trans
form(X)

def run_tsne(X,
n_components,
perplexity=30):
    tsne =
TSNE(n_components=n_
components,
perplexity=perplexity,
init='pca',
random_state=42)
    return
tsne.fit_transform(X)

X_tsne_2d =
run_tsne(X_scaled, 2,
perplexity=30)
```

```
X_tsne_3d =
run_tsne(X_scaled, 3,
perplexity=30)

plot_2d_scatter(X_tsne_2
d, y, 't-SNE 2D')
plot_3d_scatter(X_tsne_3
d, y, 't-SNE 3D')

pca_2 =
PCA(n_components=2)
X_pca_2d =
pca_2.fit_transform(X_sc
aled)

pca_3 =
PCA(n_components=3)
X_pca_3d =
pca_3.fit_transform(X_sc
aled)

plot_2d_scatter(X_pca_2
d, y, 'PCA 2D')
plot_3d_scatter(X_pca_3
d, y, 'PCA 3D')
```

Результат работы программы:



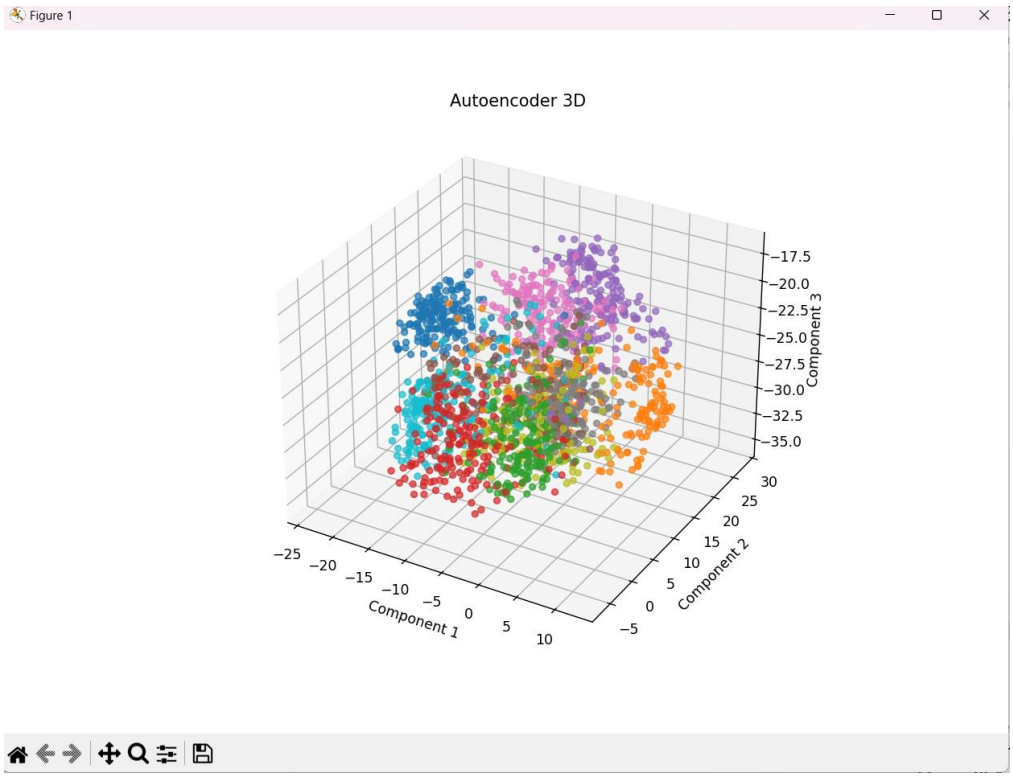


Figure 1
Autoencoder 3D
Component 1
Component 2
Component 3

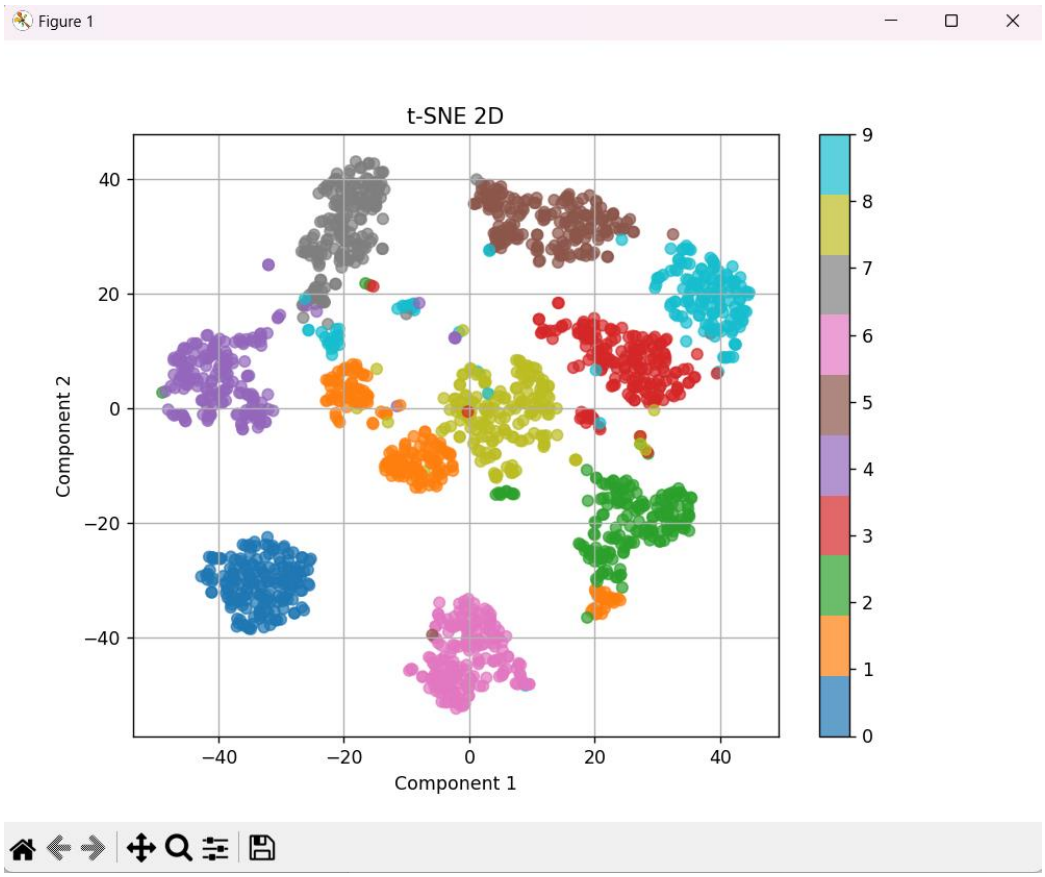


Figure 1
t-SNE 2D
Component 1
Component 2

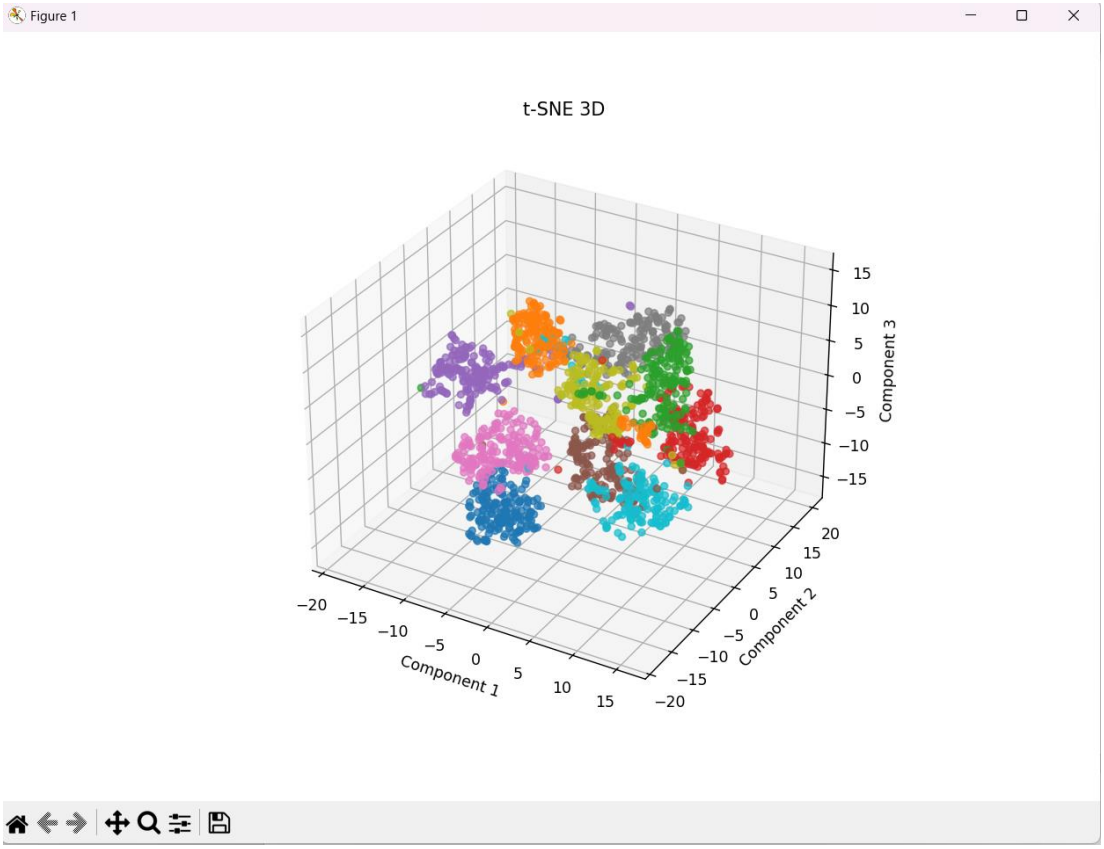


Figure 1
t-SNE 3D
Component 1
Component 2
Component 3

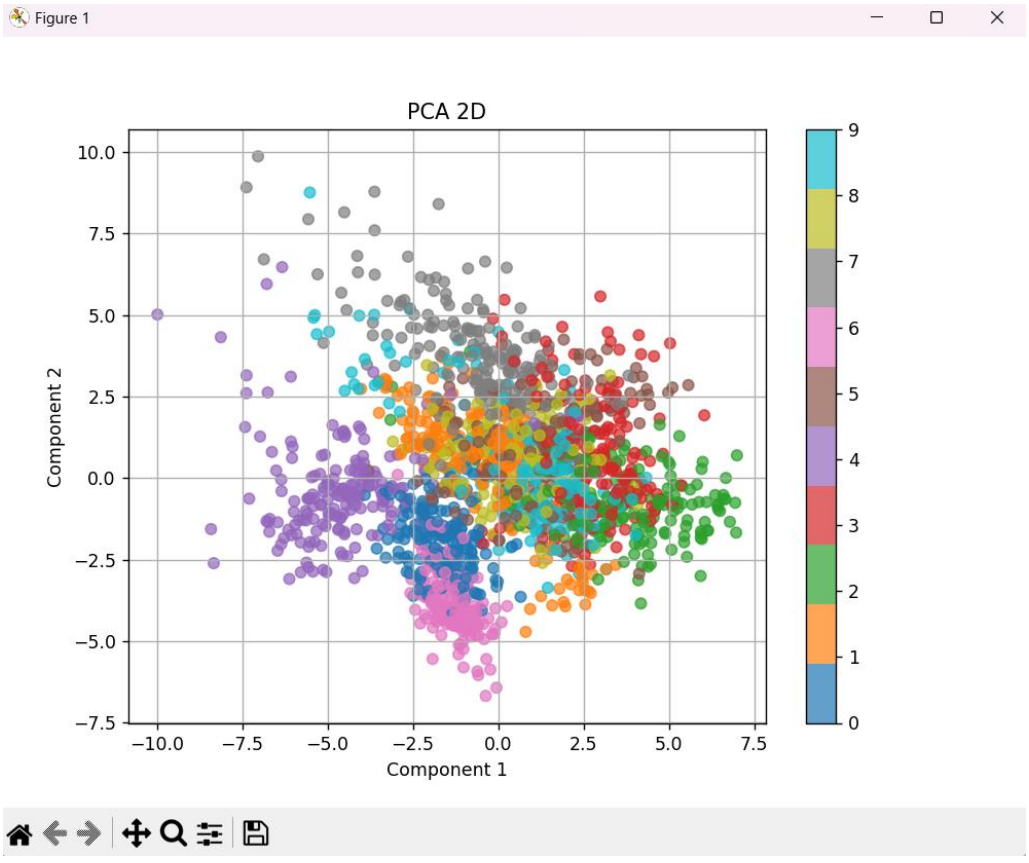
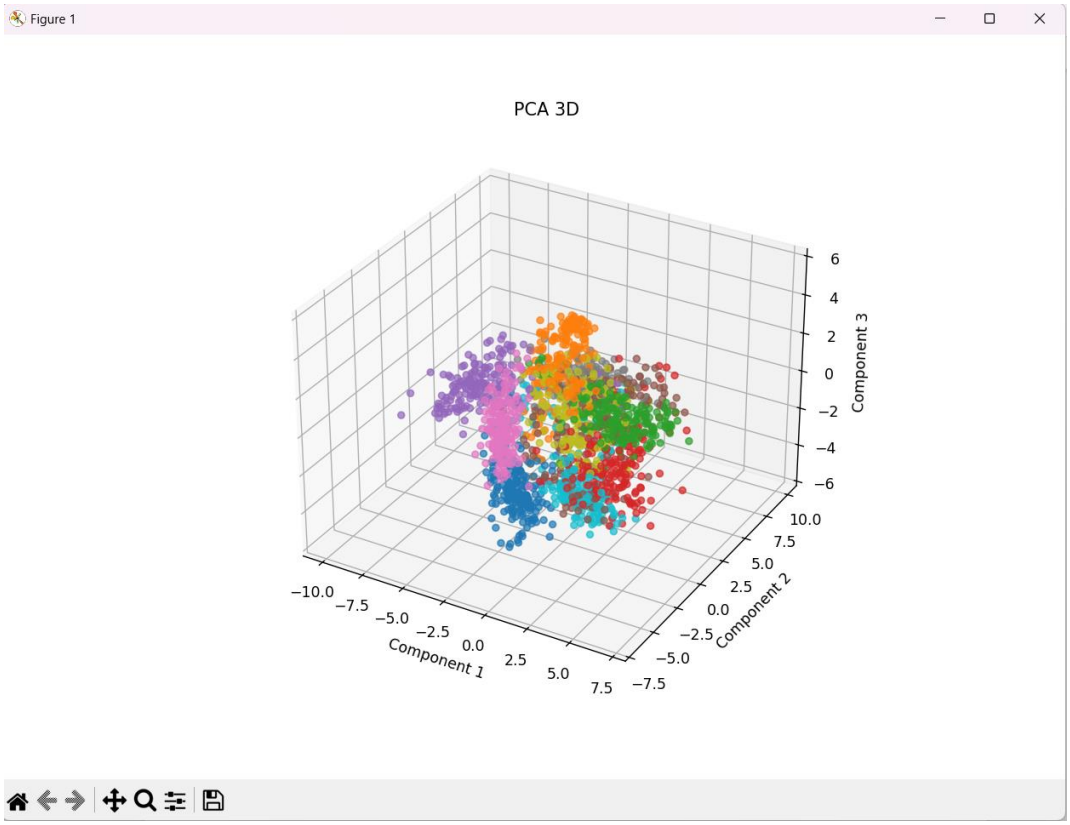


Figure 1
PCA 2D
Component 1
Component 2



По результатам визуализации видно, что метод **t-SNE** наиболее эффективно выделяет кластеры экземпляров разных классов качества. Автоэнкодер частично справляется с разделением, однако границы между классами остаются размытыми. Метод **PCA** даёт наименее выраженное разделение, так как он линейный и учитывает только максимальную дисперсию данных, а не их классовую структуру.

Вывод: научился применять автоэнкодеры для осуществления визуализации данных и их анализа.