Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

**Отчет по лабораторной работе 2**

Специальность ИИ-23

Выполнила:

Студентка 4 курса

Группы ИИ-23

Маслакова К.Ю.

Проверила:

Андреенко К.В

Преподаватель-стажер

Кафедры ИИТ,

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2025 г.

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.

**Код программы:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.manifold import TSNE

from sklearn.decomposition import PCA

import torch

from torch import nn

from torch.utils.data import DataLoader, TensorDataset

data = pd.read\_csv("winequality-white.csv", sep=';')

print("Размер данных:", data.shape)

print("Пример данных:")

print(data.head())

X = data.drop('quality', axis=1).values

y = data['quality'].values

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

X\_tensor = torch.tensor(X\_scaled, dtype=torch.float32)

y\_tensor = torch.tensor(y, dtype=torch.long)

dataset = TensorDataset(X\_tensor, y\_tensor)

loader = DataLoader(dataset, batch\_size=32, shuffle=True)

class Autoencoder(nn.Module):

def \_\_init\_\_(self, input\_dim, bottleneck\_dim):

super(Autoencoder, self).\_\_init\_\_()

self.encoder = nn.Sequential(

nn.Linear(input\_dim, 8),

nn.ReLU(),

nn.Linear(8, bottleneck\_dim)

)

self.decoder = nn.Sequential(

nn.Linear(bottleneck\_dim, 8),

nn.ReLU(),

nn.Linear(8, input\_dim)

)

def forward(self, x):

encoded = self.encoder(x)

decoded = self.decoder(encoded)

return decoded, encoded

def train\_autoencoder(model, loader, epochs=100, lr=1e-3):

optimizer = torch.optim.Adam(model.parameters(), lr=lr)

criterion = nn.MSELoss()

for epoch in range(epochs):

for batch\_X, \_ in loader:

optimizer.zero\_grad()

decoded, \_ = model(batch\_X)

loss = criterion(decoded, batch\_X)

loss.backward()

optimizer.step()

if (epoch + 1) % 20 == 0:

print(f"Эпоха [{epoch+1}/{epochs}], Потеря: {loss.item():.6f}")

autoencoder\_2 = Autoencoder(X.shape[1], 2)

autoencoder\_3 = Autoencoder(X.shape[1], 3)

print("\nОбучение автоэнкодера с 2 нейронами...")

train\_autoencoder(autoencoder\_2, loader, epochs=100)

print("\nОбучение автоэнкодера с 3 нейронами...")

train\_autoencoder(autoencoder\_3, loader, epochs=100)

with torch.no\_grad():

\_, encoded\_2 = autoencoder\_2(X\_tensor)

\_, encoded\_3 = autoencoder\_3(X\_tensor)

encoded\_2 = encoded\_2.numpy()

encoded\_3 = encoded\_3.numpy()

plt.figure(figsize=(7, 5))

scatter = plt.scatter(encoded\_2[:, 0], encoded\_2[:, 1], c=y, cmap='rainbow', alpha=0.7)

plt.colorbar(scatter, label='Класс (качество вина)')

plt.title('Автоэнкодер: 2 нейрона (2D)')

plt.xlabel('Компонента 1')

plt.ylabel('Компонента 2')

plt.show()

from mpl\_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(8, 6))

ax = fig.add\_subplot(111, projection='3d')

p = ax.scatter(encoded\_3[:, 0], encoded\_3[:, 1], encoded\_3[:, 2], c=y, cmap='rainbow', alpha=0.7)

fig.colorbar(p, ax=ax, label='Класс (качество вина)')

ax.set\_title('Автоэнкодер: 3 нейрона (3D)')

plt.show()

print("\nВыполняется t-SNE...")

tsne\_2d = TSNE(n\_components=2, random\_state=42).fit\_transform(X\_scaled)

tsne\_3d = TSNE(n\_components=3, random\_state=42).fit\_transform(X\_scaled)

plt.figure(figsize=(7, 5))

scatter = plt.scatter(tsne\_2d[:, 0], tsne\_2d[:, 1], c=y, cmap='rainbow', alpha=0.7)

plt.colorbar(scatter, label='Класс (качество вина)')

plt.title('t-SNE: 2 компоненты')

plt.show()

fig = plt.figure(figsize=(8, 6))

ax = fig.add\_subplot(111, projection='3d')

p = ax.scatter(tsne\_3d[:, 0], tsne\_3d[:, 1], tsne\_3d[:, 2], c=y, cmap='rainbow', alpha=0.7)

fig.colorbar(p, ax=ax, label='Класс (качество вина)')

ax.set\_title('t-SNE: 3 компоненты')

plt.show()

pca\_2 = PCA(n\_components=2)

pca\_3 = PCA(n\_components=3)

X\_pca\_2 = pca\_2.fit\_transform(X\_scaled)

X\_pca\_3 = pca\_3.fit\_transform(X\_scaled)

plt.figure(figsize=(7, 5))

scatter = plt.scatter(X\_pca\_2[:, 0], X\_pca\_2[:, 1], c=y, cmap='rainbow', alpha=0.7)

plt.colorbar(scatter, label='Класс (качество вина)')

plt.title('PCA: 2 компоненты')

plt.show()

fig = plt.figure(figsize=(8, 6))

ax = fig.add\_subplot(111, projection='3d')

p = ax.scatter(X\_pca\_3[:, 0], X\_pca\_3[:, 1], X\_pca\_3[:, 2], c=y, cmap='rainbow', alpha=0.7)

fig.colorbar(p, ax=ax, label='Класс (качество вина)')

ax.set\_title('PCA: 3 компоненты')

plt.show()

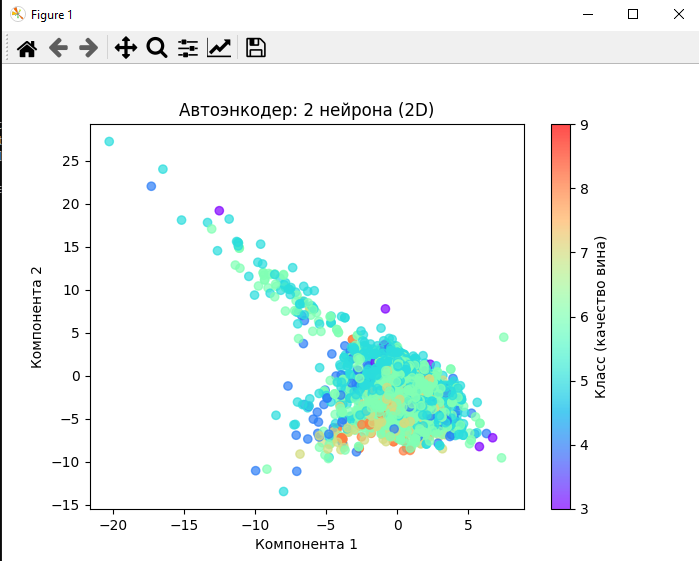
explained\_var\_2 = np.sum(pca\_2.explained\_variance\_ratio\_)

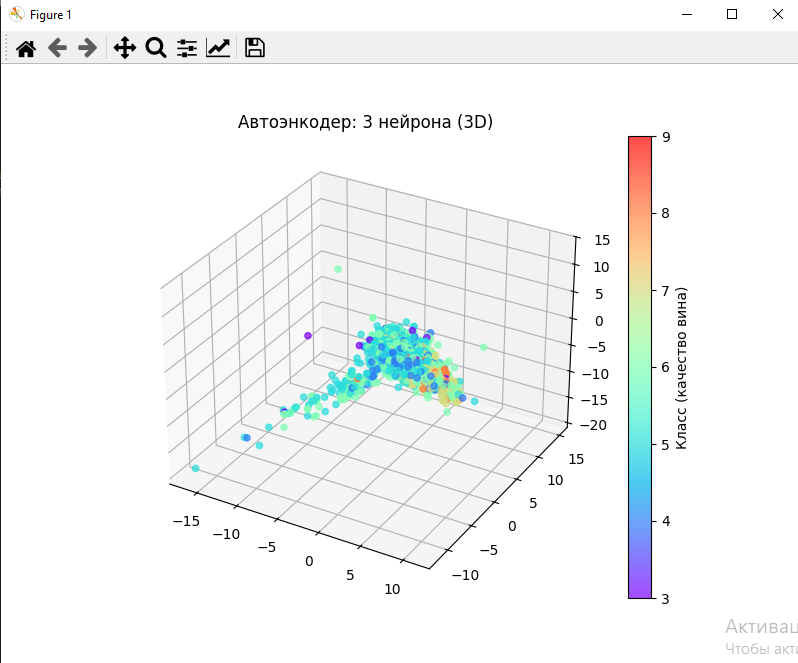
explained\_var\_3 = np.sum(pca\_3.explained\_variance\_ratio\_)

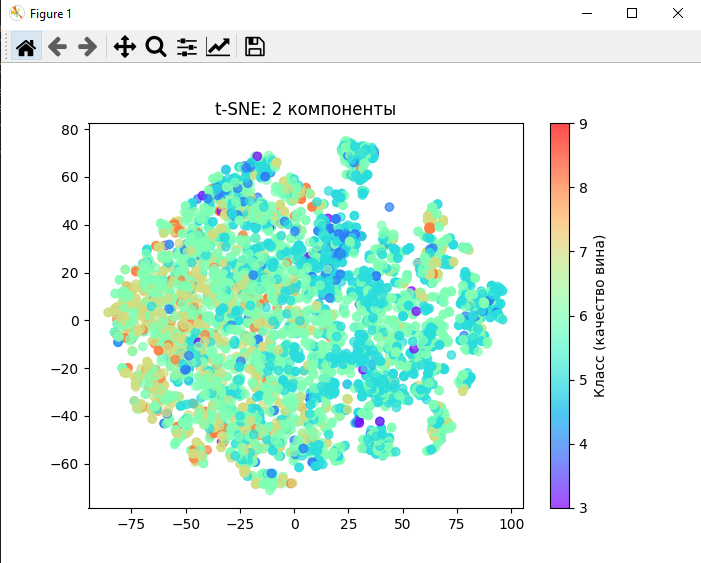
print(f"\nПотери при 2 компонентах PCA: {1 - explained\_var\_2:.4f} ({(1 - explained\_var\_2) \* 100:.2f}%)")

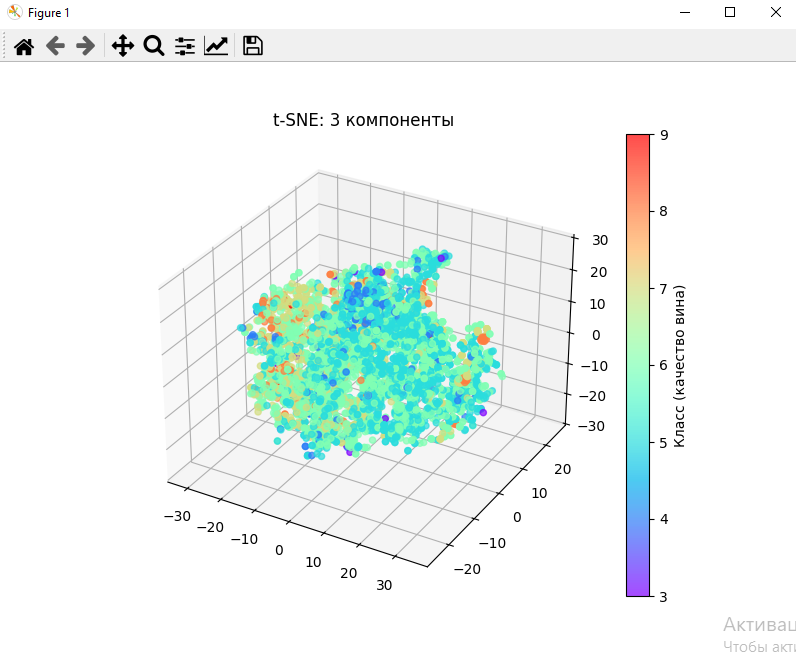
print(f"Потери при 3 компонентах PCA: {1 - explained\_var\_3:.4f} ({(1 - explained\_var\_3) \* 100:.2f}%)")

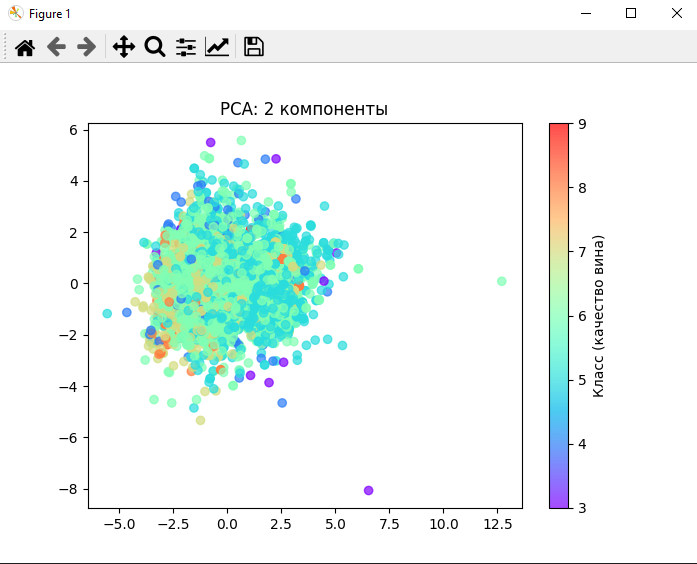
**Результат работы программы:**

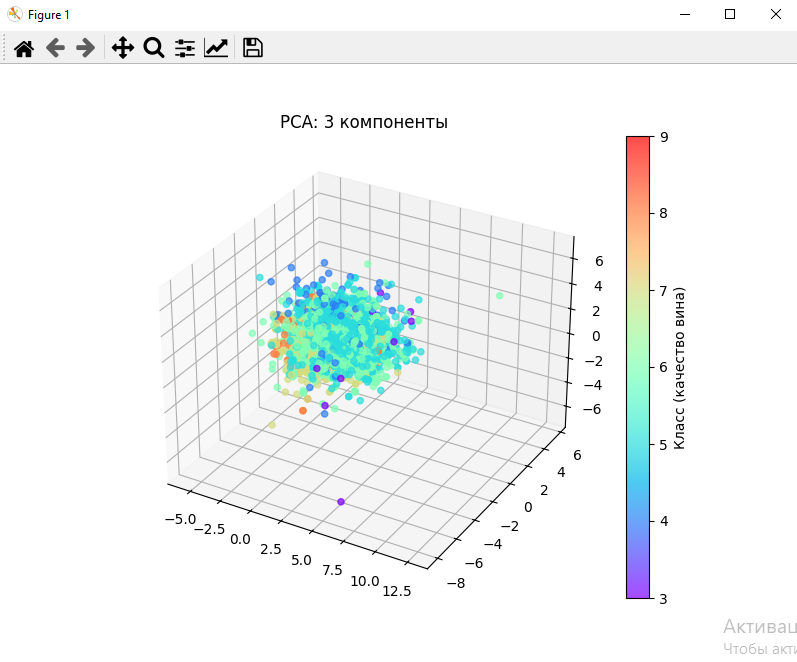
****

****

****

****

****

****

Вывод: научилась применять автоэнкодеры для осуществления визуализации данных и их анализа