`

Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Отчет по лабораторной работе 2

Специальность ИИ-23

Выполнил:

Гавришук В.Р.

Студент группы ИИ-23

Проверил:

Андренко К. В.
Преподаватель-стажёр
Кафедры ИИТ,
«___» _____2025 г.

Брест 2025

`

Лабораторная работа № 2. Автоэнкодеры

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

№ варианта	Выборка	Класс
4	Wine Quality (white)	quality

Ход работы:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.model_selection import train_test_split
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader
import warnings

warnings.filterwarnings('ignore')

path = "winequality-white.csv"
df = pd.read_csv(path, sep=';')
X = df.drop(columns=['quality']).values.astype(np.float32)
y = df['quality'].values
scaler = StandardScaler()
Xs = scaler.fit_transform(X).astype(np.float32)
```

`

```
input_dim = Xs.shape[1]
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print("Используем устройство:", device)


class Autoencoder(nn.Module):
    def __init__(self, input_dim, bottleneck_dim):
        super().__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, bottleneck_dim)
        )
        self.decoder = nn.Sequential(
            nn.Linear(bottleneck_dim, 32),
            nn.ReLU(),
            nn.Linear(32, 64),
            nn.ReLU(),
            nn.Linear(64, input_dim)
        )

    def forward(self, x):
        z = self.encoder(x)
        xrec = self.decoder(z)
        return xrec

    def encode(self, x):
        with torch.no_grad():
            return self.encoder(x)


def train_ae(X, bottleneck_dim, epochs=200, batch_size=32, lr=1e-3, verbose=False):
    X_train, X_val = train_test_split(X, test_size=0.1, random_state=42)
    train_ds = TensorDataset(torch.from_numpy(X_train))
    val_ds = TensorDataset(torch.from_numpy(X_val))
    train_loader = DataLoader(train_ds, batch_size=batch_size, shuffle=True)
    val_loader = DataLoader(val_ds, batch_size=batch_size, shuffle=False)

    model = Autoencoder(input_dim, bottleneck_dim).to(device)
    opt = torch.optim.Adam(model.parameters(), lr=lr)
    loss_fn = nn.MSELoss()

    best_val = np.inf
    best_state = None
    patience = 12
    wait = 0

    for ep in range(epochs):
        model.train()
        train_loss = 0.0
        for batch in train_loader:
            xb = batch[0].to(device)
            opt.zero_grad()
            xr = model(xb)
            loss = loss_fn(xr, xb)
            loss.backward()
            opt.step()
            train_loss += loss.item() * xb.size(0)
        train_loss /= len(train_loader.dataset)


        model.eval()
        val_loss = 0.0
        with torch.no_grad():
```

`

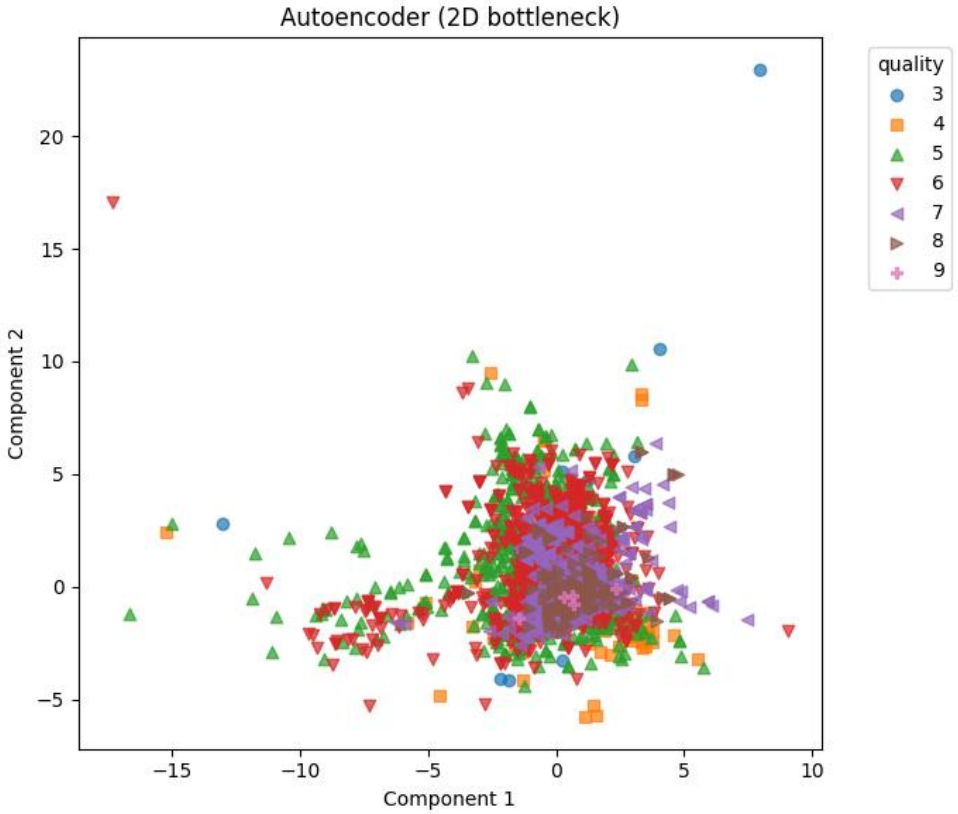
```
            for batch in val_loader:
                xb = batch[0].to(device)
                xr = model(xb)
                l = loss_fn(xr, xb)
                val_loss += l.item() * xb.size(0)
        val_loss /= len(val_loader.dataset)

        if verbose and (ep % 10 == 0 or ep == epochs - 1):
            print(f"Epoch {ep + 1}/{epochs} train_loss={train_loss:.6f} val_loss={val_loss:.6f}")


        if val_loss < best_val - 1e-6:
            best_val = val_loss
            best_state = {k: v.cpu().clone() for k, v in model.state_dict().items()}
            wait = 0
        else:
            wait += 1
            if wait >= patience:
                if verbose:
                    print(f"Early stopping at epoch {ep + 1}, best_val={best_val:.6f}")
                break

    if best_state is not None:
        model.load_state_dict(best_state)
    return model


model_ae2 = train_ae(Xs, 2, epochs=200, batch_size=32, lr=1e-3, verbose=True)
model_ae3 = train_ae(Xs, 3, epochs=200, batch_size=32, lr=1e-3, verbose=True)


Xs_torch = torch.from_numpy(Xs).to(device)
with torch.no_grad():
    Z_ae2 = model_ae2.encode(Xs_torch).cpu().numpy()
    Z_ae3 = model_ae3.encode(Xs_torch).cpu().numpy()

print("Размеры латентных представлений:", Z_ae2.shape, Z_ae3.shape)


classes = np.unique(y)
marker_list = ['o', 's', '^', 'v', '<', '>', 'P', 'X', 'D', '*', '+']
markers = {cls: marker_list[i % len(marker_list)] for i, cls in enumerate(classes)}

def plot_2d(X2, labels, title):
    plt.figure(figsize=(7, 6))
    for cls in classes:
        mask = labels == cls
        plt.scatter(X2[mask, 0], X2[mask, 1], label=str(cls), marker=markers[cls], alpha=0.7)
    plt.legend(title='quality', bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.title(title)
    plt.xlabel('Component 1')
    plt.ylabel('Component 2')
    plt.tight_layout()
    plt.show()

def plot_3d(X3, labels, title):
    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(111, projection='3d')
    for cls in classes:
        mask = labels == cls
        ax.scatter(X3[mask, 0], X3[mask, 1], X3[mask, 2], label=str(cls), marker=markers[cls], alpha=0.7)
    ax.set_title(title)
    ax.set_xlabel('Component 1')
    ax.set_ylabel('Component 2')
    ax.set_zlabel('Component 3')
    ax.legend(title='quality', bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.tight_layout()
    plt.show()
```

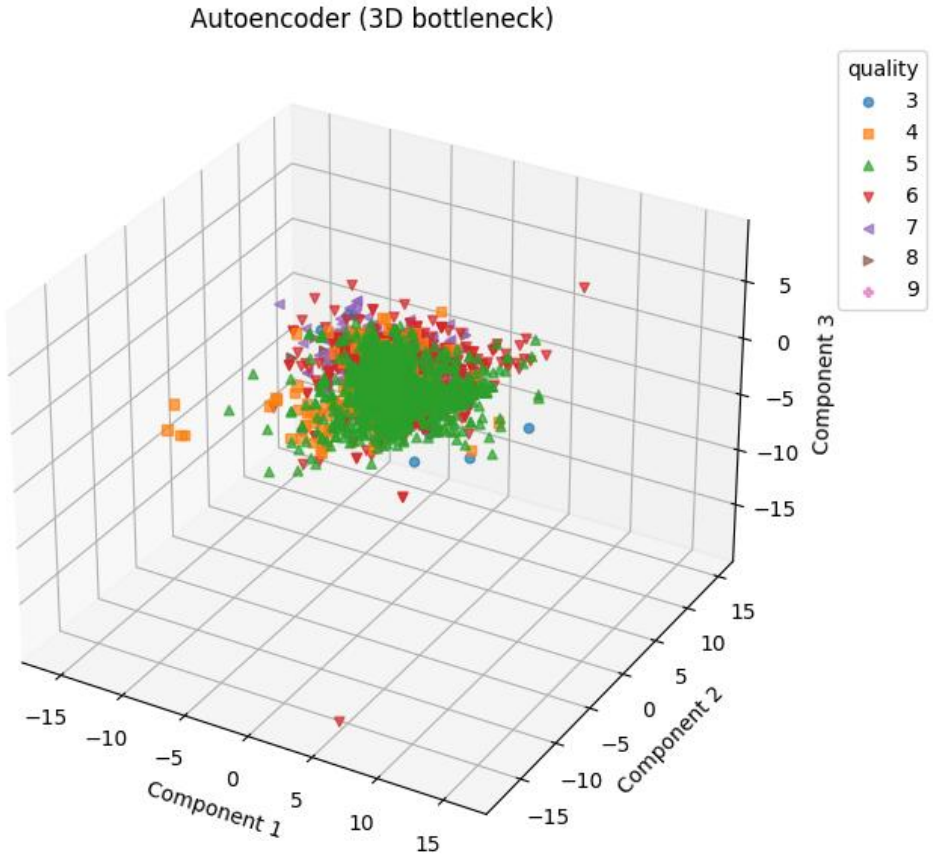
`

```
plot_2d(Z_ae2, y, "Autoencoder (2D bottleneck)")
plot_3d(Z_ae3, y, "Autoencoder (3D bottleneck)")


perps = [20, 30, 50]
tsne_2d_results = {}
for p in perps:
    tsne = TSNE(n_components=2, perplexity=p, init='pca', random_state=42, learning_rate='auto')
    Z = tsne.fit_transform(Xs)
    tsne_2d_results[p] = Z
    plot_2d(Z, y, f"t-SNE 2D (perplexity={p})")

chosen_p = 30
Z_tsne2_chosen = tsne_2d_results[chosen_p]
tsne3 = TSNE(n_components=3, perplexity=chosen_p, init='pca', random_state=42, learning_rate='auto')
Z_tsne3 = tsne3.fit_transform(Xs)
plot_3d(Z_tsne3, y, f"t-SNE 3D (perplexity={chosen_p})")


pca2 = PCA(n_components=2)
Z_pca2 = pca2.fit_transform(Xs)
plot_2d(Z_pca2, y, "PCA (2 components)")

pca3 = PCA(n_components=3)
Z_pca3 = pca3.fit_transform(Xs)
plot_3d(Z_pca3, y, "PCA (3 components)")


out_df = pd.DataFrame({
    'ae2_1': Z_ae2[:, 0],
    'ae2_2': Z_ae2[:, 1],
    'ae3_1': Z_ae3[:, 0],
    'ae3_2': Z_ae3[:, 1],
    'ae3_3': Z_ae3[:, 2],
    'tsne2_30_1': Z_tsne2_chosen[:, 0],
    'tsne2_30_2': Z_tsne2_chosen[:, 1],
    'tsne3_30_1': Z_tsne3[:, 0],
    'tsne3_30_2': Z_tsne3[:, 1],
    'tsne3_30_3': Z_tsne3[:, 2],
    'pca2_1': Z_pca2[:, 0],
    'pca2_2': Z_pca2[:, 1],
    'pca3_1': Z_pca3[:, 0],
    'pca3_2': Z_pca3[:, 1],
    'pca3_3': Z_pca3[:, 2],
    'quality': y
})

print("\nПервые строки результирующего DataFrame:")
print(out_df.head())

out_df.to_csv("wine_representations.csv", index=False)
print("\nФайл 'wine_representations.csv' сохранён в рабочей директории.")
print("Готово. Построены визуализации AE, t-SNE и PCA.")
```

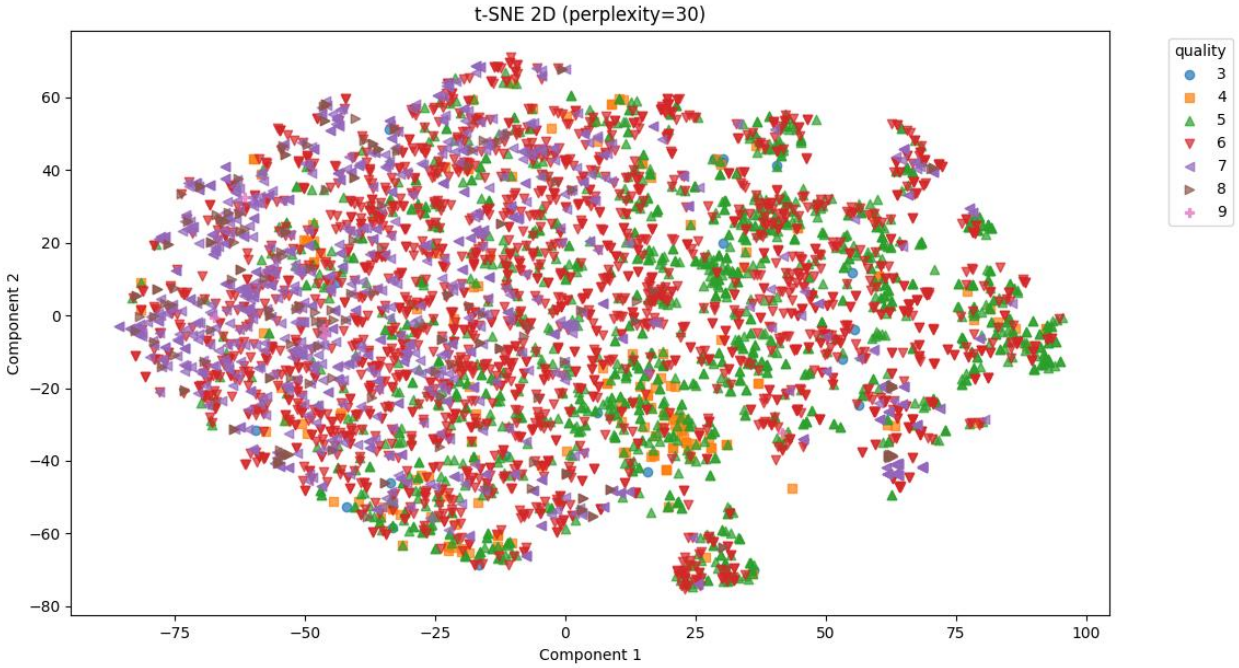
`

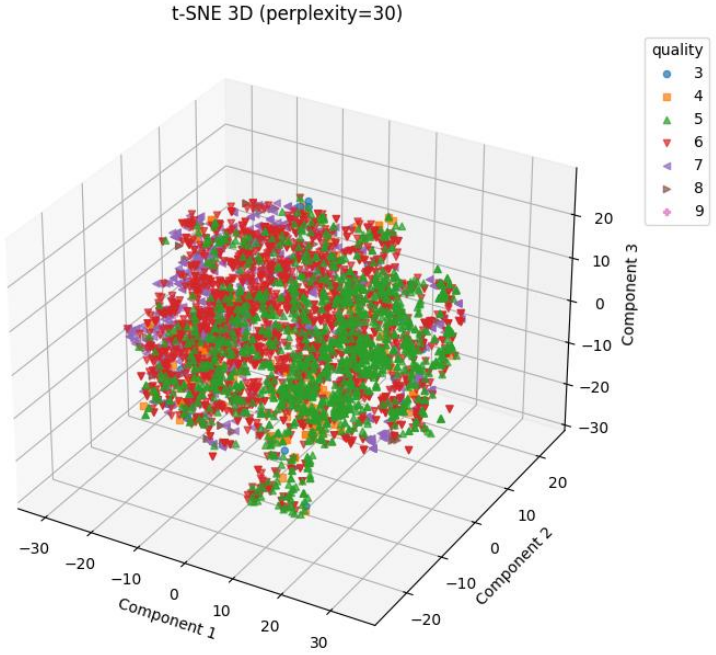
Результат работы программы:



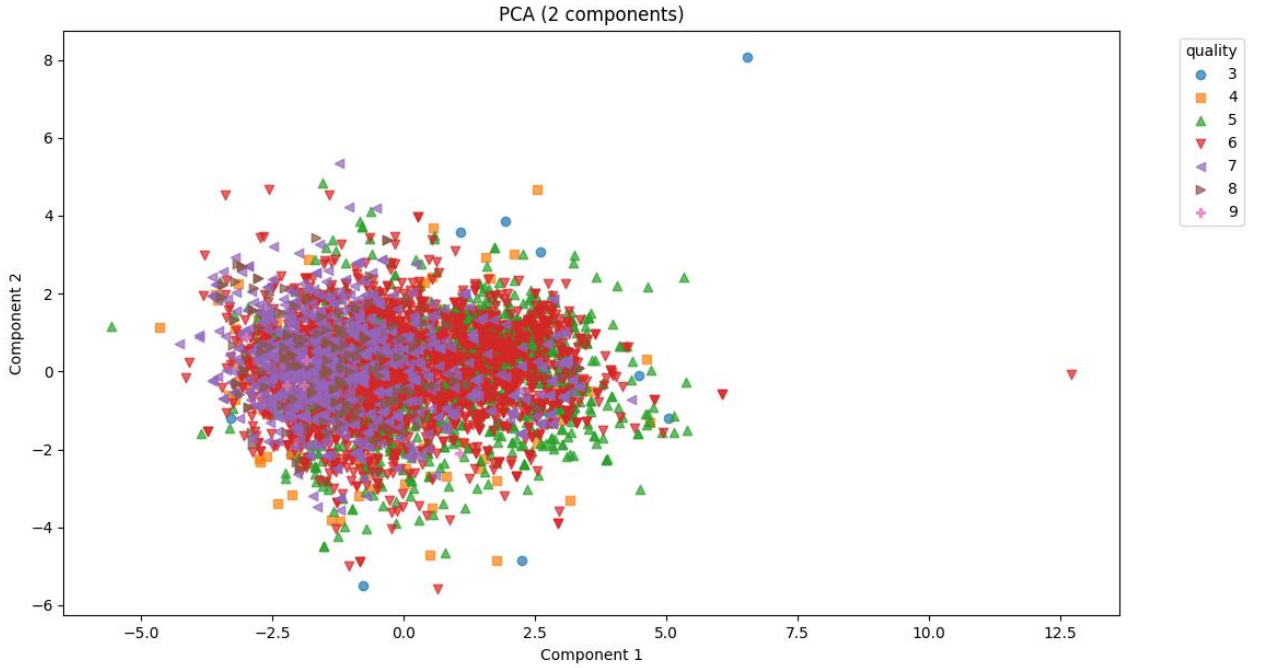


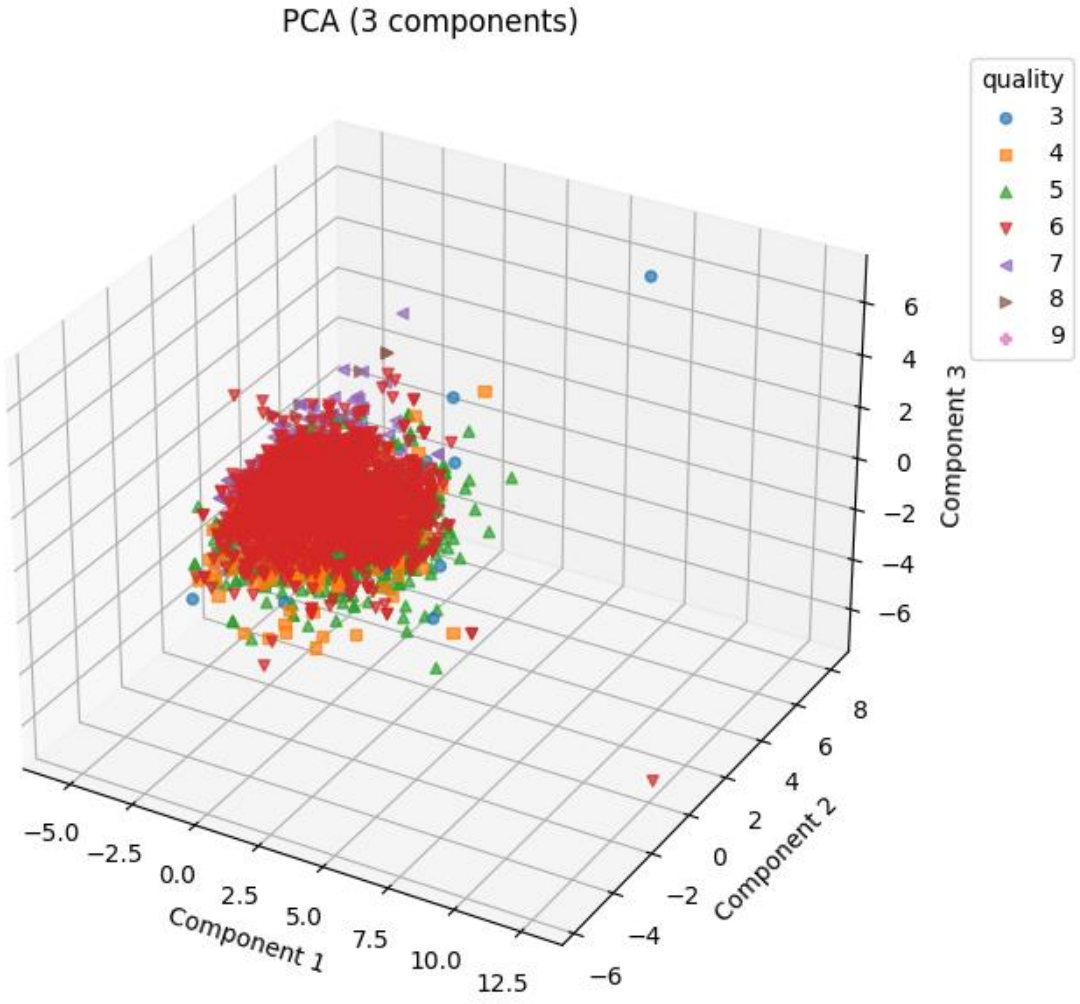
`


t-SNE 2D (perplexity=30)
quality
3
4
5
6
7
8
9
Component 2
Component 1
60
40
20
0
−20
−40
−60
−80
−75
−50
−25
0
25
50
75
100


t-SNE 3D (perplexity=30)
quality
3
4
5
6
7
8
9
Component 1
Component 2
Component 3
−30
−20
−10
0
10
20
30
20
10
0
−10
−20
−30

`





Автоэнкодер хорошо справился с задачей снижения размерности, сохранив структуре данных.

t-SNE показал лучшее визаульное разделение классов, это подтверждает наличие нелинейной структуры в данных.

PCA дает наглядную аппроксимацию структуры данных, но при высокой нелинейности признаков теряет способность различать классы.

`

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