Министерство образования Республики Беларусь
Учреждение образования
“Брестский государственный технический университет”
Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №2
Специальность ИИ-22

Выполнил:
А. А. Сидоренко
Студент группы ИИ-22

Проверил:
А. А. Крощенко
доц. кафедры ИИТ

Брест 2024

Цель: научиться конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения.

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;

Ход работы:

Вариант 16

Выборка: seeds.txt

Код:

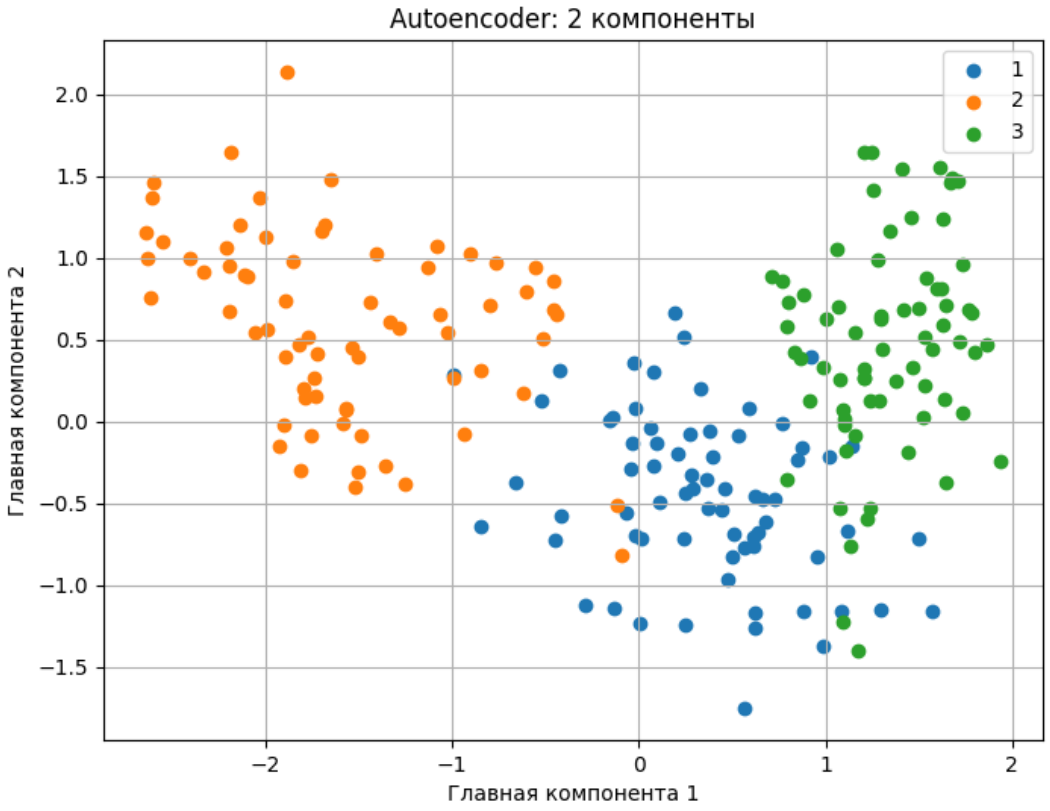
```
import pandas as pd
import numpy as np
import torch.nn as nn
import torch.optim as optim
from tqdm import tqdm
import torch
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler


def get_data():
    data = pd.read_csv("seeds_dataset.txt", delim_whitespace=True,
header=None)
    feature = data.iloc[:, :-1].values
    scaler = StandardScaler()
    normalized_features = scaler.fit_transform(feature)
    target = data.iloc[:, -1].values
    return normalized_features, target


class Autoencoder(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 256),
            nn.ReLU(),
            nn.Linear(256, hidden_dim)
        )
        self.decoder = nn.Sequential(
            nn.Linear(hidden_dim, 256),
            nn.ReLU(),
            nn.Linear(256, input_dim)
```

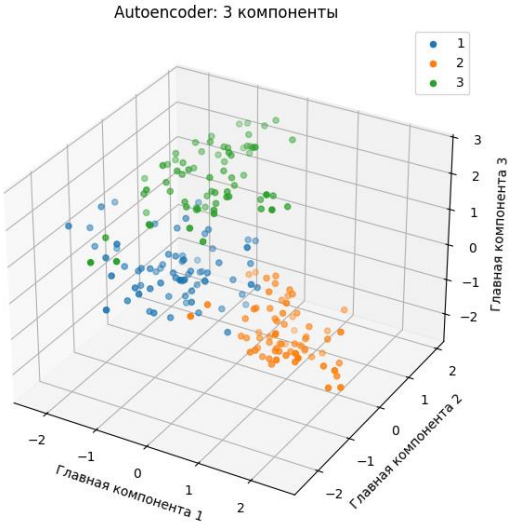
```
        )

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


def train(autoencoder, data, epochs=50, lr=0.001):
    optimizer = optim.Adam(autoencoder.parameters(), lr=lr)
    criterion = nn.MSELoss()

    with tqdm(range(epochs), desc="Training Progress") as pbar:
        for epoch in pbar:
            data_tensor = torch.FloatTensor(data)

            encoded, decoded = autoencoder(data_tensor)

            loss = criterion(decoded, data_tensor)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            pbar.set_postfix({"Loss": f"{loss.item():.4f}"})

    return autoencoder


def visualize(data, labels, components, title):
    classes = np.unique(labels)

    if components == 2:
        plt.figure(figsize=(8, 6))
        for cls in classes:
            cls_idx = labels == cls
            plt.scatter(data[cls_idx, 0], data[cls_idx, 1], label=cls)

        plt.title(title)
        plt.xlabel('Главная компонента 1')
        plt.ylabel('Главная компонента 2')
        plt.legend()
        plt.grid(True)
        plt.show()

    elif components == 3:
        fig = plt.figure(figsize=(10, 8))
        ax = fig.add_subplot(111, projection='3d')
        for cls in classes:
            cls_idx = labels == cls
            ax.scatter(data[cls_idx, 0], data[cls_idx, 1], data[cls_idx,
2], label=cls)

        ax.set_title(title)
        ax.set_xlabel('Главная компонента 1')
        ax.set_ylabel('Главная компонента 2')
        ax.set_zlabel('Главная компонента 3')
        ax.legend()
        plt.show()


def tsne(data, labels, components, perplexity=100):
    tsne = TSNE(n_components=components, perplexity=perplexity,
init='random', random_state=0)
    tsne_transformed = tsne.fit_transform(data)
    visualize(tsne_transformed, labels, components, title=f't-SNE:
```

```
{components} компоненты')


def main():
    features, targets = get_data()
    nn_2c = Autoencoder(features.shape[1], 2)
    nn_2c_trd = train(nn_2c, features)
    c2, _ = nn_2c_trd(torch.FloatTensor(features))
    c2 = c2.detach().numpy()
    visualize(c2, targets, 2, title='Autoencoder: 2 компоненты')

    nn_3c = Autoencoder(features.shape[1], 3)
    nn_3c_trd = train(nn_3c, features)
    c3, _ = nn_3c_trd(torch.FloatTensor(features))
    c3 = c3.detach().numpy()
    visualize(c3, targets, 3, title='Autoencoder: 3 компоненты')

    tsne(features, targets, components=2, perplexity=40)
    tsne(features, targets, components=3, perplexity=40)


if __name__ == '__main__':
    main()
```

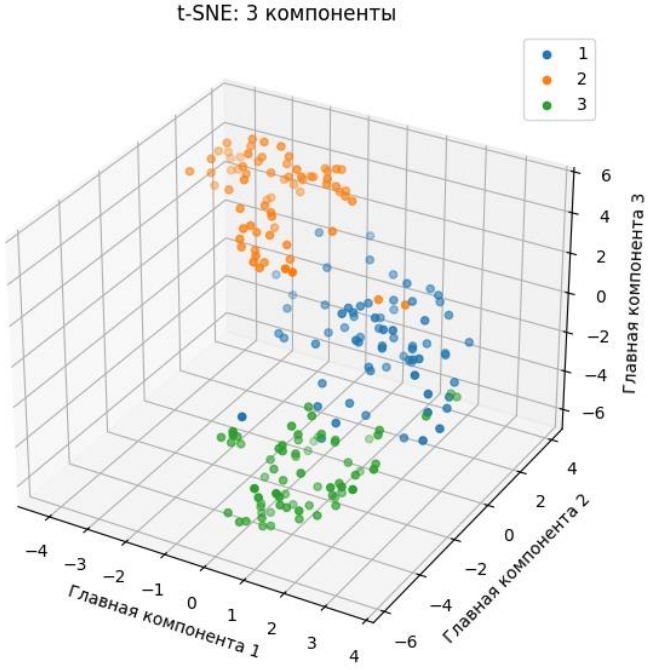
```
Training Progress: 100%|██████████| 50/50 [00:00<00:00, 450.61it/s, Loss=0.1215]
Training Progress: 100%|██████████| 50/50 [00:00<00:00, 294.12it/s, Loss=0.0507]
```

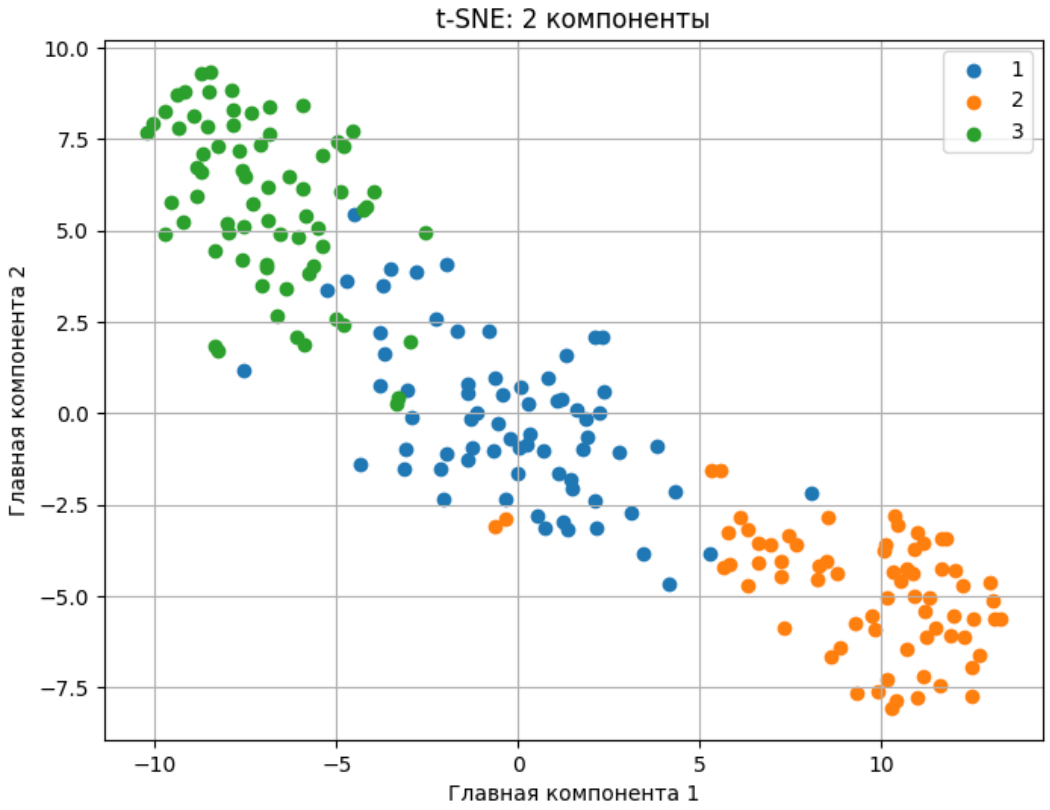
Проецирование данных на плоскость первых двух и трех главных компонент с использованием нейросетевой модели автоэнкодера





Визуализация данных для метода t-SNE





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