Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №1**

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «PCA и автоэнкодеры»

**Выполнил:**

Студент 3 курса

Группы ИИ-21

Карагодин Д. Л.

**Проверил:**

Туз И. С.

Брест 2024

**Цель:** необходимо реализовать PCA и автоэнкодер и сравнить время и качество их обучения.

**Ход работы**

**Вариант 1**

1. mushrooms

**Код программы (PCA):**

import numpy as np

import pandas as pd

import time

import matplotlib.pyplot as plt

# Загрузка данных из файла CSV

data = pd.read\_csv("Seed\_Data.csv")

# Извлечение вещественных атрибутов

attributes = data.iloc[:, :-1].values

# Среднее значение для центрирования данных

mean\_values = np.mean(attributes, axis=0)

# Центрирование данных

centered\_data = attributes - mean\_values

start\_time = time.time()

# Ковариационная матрица

covariance\_matrix = np.cov(centered\_data, rowvar=False)

# Вычисление собственных значений и собственных векторов

eigenvalues, eigenvectors = np.linalg.eigh(covariance\_matrix)

end\_time = time.time()

# Вывод времени выполнения

print(f"Время выполнения PCA: {end\_time - start\_time:.5f} секунд")

# Сортировка собственных значений и векторов в убывающем порядке

sorted\_indices = np.argsort(eigenvalues)[::-1]

eigenvalues = eigenvalues[sorted\_indices]

eigenvectors = eigenvectors[:, sorted\_indices]

# Количество компонент для сохранения

num\_components = 2

# Выбор заданного количества компонент

selected\_components = eigenvectors[:, :num\_components]

# Преобразование данных в новое пространство

transformed\_data = np.dot(centered\_data, selected\_components)

# Восстановление данных из преобразованных данных

reconstructed\_data = np.dot(transformed\_data, selected\_components.T) + mean\_values

# Расчет среднеквадратичного отклонения (MSE)

mse = np.mean((attributes - reconstructed\_data)\*\*2)

# Вывод ошибки

print(f"\nСреднеквадратичное отклонение (MSE) между оригинальными и восстановленными данными: {mse:.5f}")

# Вывод результата

print("\nПреобразованные данные после PCA:")

print(pd.DataFrame(transformed\_data, columns=[f"PC{i+1}" for i in range(num\_components)]))

# График данных и собственных векторов

plt.scatter(transformed\_data[:, 0], transformed\_data[:, 1], alpha=0.7)

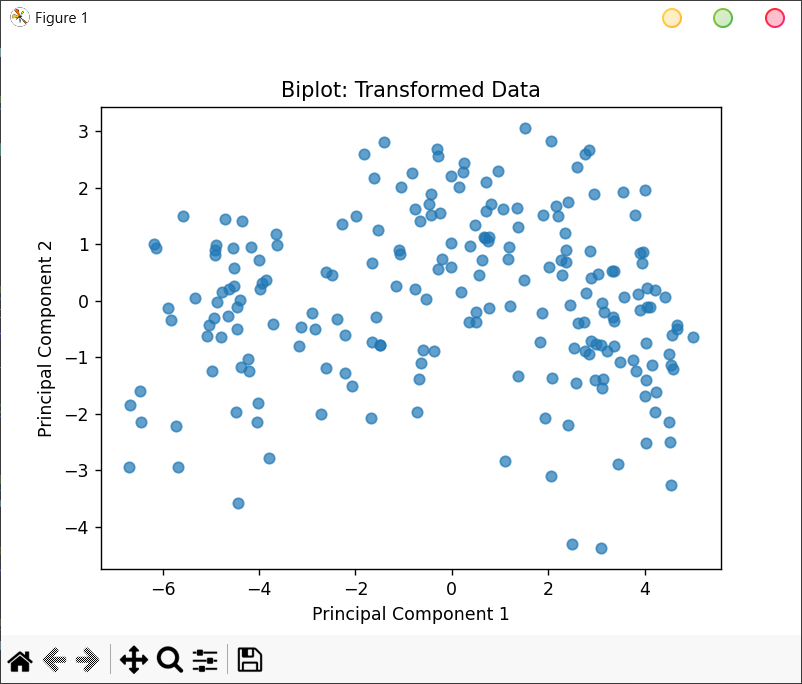
plt.title('Biplot: Transformed Data')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.show()

**Результат программы (PCA):**



Время выполнения PCA: 0.00401 секунд

Среднеквадратичное отклонение (MSE) между оригинальными и восстановленными данными: 0.01292

Преобразованные данные после PCA:

PC1 PC2

0 -0.663448 1.417321

1 -0.315667 2.689229

2 0.660499 1.131506

3 1.055276 1.621190

4 -1.619999 2.183384

.. ... ...

205 3.041872 0.473126

206 4.109063 -0.109341

207 2.500034 -4.307965

208 3.332079 0.525290

209 3.107551 -1.549757

[210 rows x 2 columns]

**Код программы (Автоэнкодер):**

import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import DataLoader, TensorDataset

from sklearn.preprocessing import StandardScaler

import pandas as pd

import time

import matplotlib.pyplot as plt  # Add this line for matplotlib

# Загрузка данных из файла CSV

data = pd.read\_csv("Seed\_Data.csv")

# Извлечение вещественных атрибутов

attributes = data.iloc[:, :-1].values

# Нормализация данных

scaler = StandardScaler()

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

# Преобразование в Tensor

tensor\_data = torch.Tensor(attributes\_scaled)

# Класс автоэнкодера

class Autoencoder(nn.Module):

    def \_\_init\_\_(self, input\_size, encoding\_size):

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

        self.encoder = nn.Linear(input\_size, encoding\_size)

        self.decoder = nn.Linear(encoding\_size, input\_size)

    def forward(self, x):

        x = self.encoder(x)

        x = self.decoder(x)

        return x

# Параметры автоэнкодера

input\_size = attributes.shape[1]

encoding\_size = 2

# Создание модели и оптимизатора

autoencoder = Autoencoder(input\_size, encoding\_size)

optimizer = optim.Adam(autoencoder.parameters(), lr=0.001)

criterion = nn.MSELoss()

# Создание DataLoader

batch\_size = 32

dataset = TensorDataset(tensor\_data, tensor\_data)

dataloader = DataLoader(dataset, batch\_size=batch\_size, shuffle=True)

# Обучение автоэнкодера

start\_time = time.time()

num\_epochs = 100

for epoch in range(num\_epochs):

    for batch in dataloader:

        data\_batch, \_ = batch

        optimizer.zero\_grad()

        outputs = autoencoder(data\_batch)

        loss = criterion(outputs, data\_batch)

        loss.backward()

        optimizer.step()

end\_time = time.time()

# Вывод времени обучения

print(f"Время обучения автоэнкодера: {end\_time - start\_time:.5f} секунд")

autoencoder.eval()

with torch.no\_grad():

    encoded\_data = autoencoder.encoder(tensor\_data)

encoded\_data\_np = encoded\_data.numpy()

plt.scatter(encoded\_data\_np[:, 0], encoded\_data\_np[:, 1], c='b', marker='o', label='Encoded Data')

plt.title('Autoencoder Latent Space')

plt.xlabel('Latent Dimension 1')

plt.ylabel('Latent Dimension 2')

plt.legend()

plt.show()

# Вычисление ошибки между входными и восстановленными данными

with torch.no\_grad():

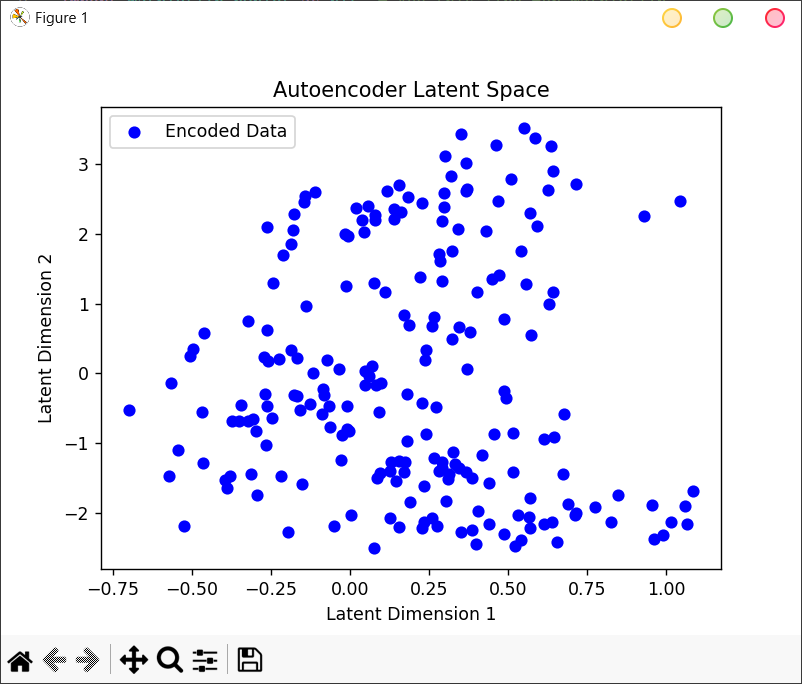
    reconstructed\_data = autoencoder(tensor\_data)

    reconstruction\_error = criterion(reconstructed\_data, tensor\_data)

# Вывод ошибки

print(f"Ошибка восстановления данных: {reconstruction\_error.item():.5f}")

**Результат программы (Автоэнкодер):**



Время обучения автоэнкодера: 0.59908 секунд

Ошибка восстановления данных: 0.23414

**Цель:** реализовал PCA и автоэнкодер и сравнить время и качество их обучения.