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

Литвинюк Т. В.

**Проверил:**

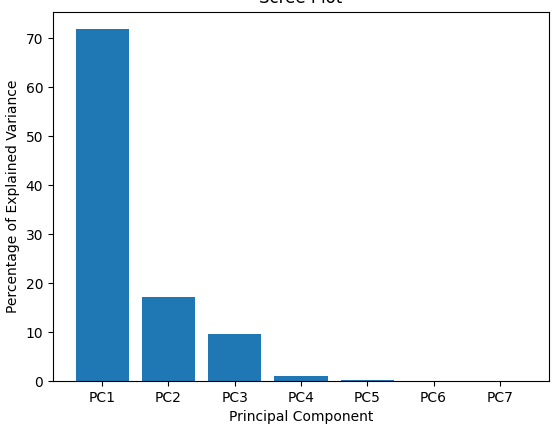
Туз И. С.

Брест 2024

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

**Ход работы**

**Вариант 7**

**РСА:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

class PCA:

    def standardize\_data(*self*, *X*):

        mean = np.mean(X, *axis*=0)

        std = np.std(X, *axis*=0)

        scaled\_X = (X - mean) / std

        return scaled\_X, mean, std

    def fit(*self*, *X*, *n\_components*):

        # Стандартизация данных

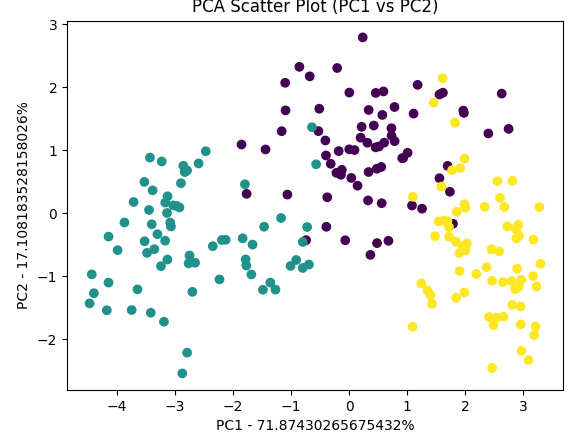
        scaled\_X, self.mean, self.std = self.standardize\_data(X)

        # Вычисление матрицы ковариации

        covariance\_matrix = np.cov(scaled\_X.T)

        # Вычисление собственных векторов и собственных значений

        eigen\_values, eigen\_vectors = np.linalg.eig(covariance\_matrix)

****        # Сортировка собственных значений и векторов

        eigen\_pairs = [(np.abs(eigen\_values[i]), eigen\_vectors[:,i]) for i in range(len(eigen\_values))]

        eigen\_pairs.sort(*key*=lambda *x*: x[0], *reverse*=True)

        # Выбор компонент для проекции

        self.components = np.hstack([eigen\_pairs[i][1].reshape(-1,1) for i in range(n\_components)])

        # Вычисление объясненной дисперсии

        total\_variance = np.sum(eigen\_values)

        self.explained\_variance = [(i / total\_variance) \* 100 for i in eigen\_values]

    def transform(*self*, *X*):

        # Проецирование данных на главные компоненты

        return np.dot((X - self.mean) / self.std, self.components)

    def inverse\_transform(*self*, *X\_transformed*):

        # Обратное преобразование PCA

        return np.dot(X\_transformed, self.components.T) \* self.std + self.mean

****

**Autoencoder:**

import torch

import torch.nn as nn

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

class Autoencoder(nn.Module):

    def \_\_init\_\_(*self*):

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

        self.encoder = nn.Sequential(

            nn.Linear(7, 4),

            nn.ReLU(),

            nn.Linear(4, 2),

            nn.ReLU(),

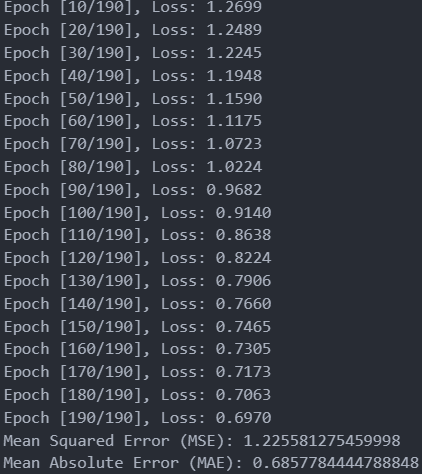
        )

        self.decoder = nn.Sequential(

            nn.Linear(2, 4),

            nn.ReLU(),

            nn.Linear(4, 7),

****            nn.Sigmoid()  # Применяем сигмоид для ограничения значений в диапазоне [0, 1]

        )

    def forward(*self*, *x*):

        x = self.encoder(x)

        x = self.decoder(x)

        return x

data = pd.read\_csv("Seed\_Data.csv", *header*=None)

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

scaler = StandardScaler()

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

# Конвертация данных в тензоры PyTorch

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

model = Autoencoder()

criterion = nn.MSELoss()

optimizer = torch.optim.Adam(model.parameters(), *lr*=0.003)

num\_epochs = 190

for epoch in range(num\_epochs):

    model.train()

    optimizer.zero\_grad()

    outputs = model(X\_tensor)

    loss = criterion(outputs, X\_tensor)

    loss.backward()

    optimizer.step()

    if (epoch+1) % 10 == 0:

        print(f'Epoch [{epoch+1}/{num\_epochs}], Loss: {loss.item():.4f}')

# Применение модели для восстановления данных

model.eval()

with torch.no\_grad():

    decoded\_data = model(X\_tensor)

# Восстановление данных и их обратное масштабирование

decoded\_data = decoded\_data.numpy()

decoded\_data = scaler.inverse\_transform(decoded\_data)

mse = mean\_squared\_error(X, decoded\_data)

# Вычисление средней абсолютной ошибки

mae = mean\_absolute\_error(X, decoded\_data)

print("Mean Squared Error (MSE):", mse)

print("Mean Absolute Error (MAE):", mae)

Таким образом, РСА отрабатывает точнее, чем автоенкодер.

**Вывод:** в ходе лабораторной работы я реализовал PCA и автоэнкодер и сравнил время и качество их обучения.