Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

**Отчет по лабораторной работе 3**

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 осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода |

**Общее задание**

1. Взять за основу любую сверточную или полносвязную архитектуру с количеством слоев более 3. Осуществить ее обучение (без предобучения) в соответствии с вариантом задания. Получить оценку эффективности модели, используя метрики, специфичные для решаемой задачи (например, MAPE – для регрессионной задачи или F1/Confusion matrix для классификационной).

2. Выполнить обучение с предобучением, используя автоэнкодерный подход, алгоритм которого изложен в лекции. Условие останова (например, по количеству эпох) при обучении отдельных слоев с использованием автоэнкодера выбрать самостоятельно.

3. Сравнить результаты, полученные при обучении с/без предобучения, сделать выводы.

4. Выполните пункты 1-3 для датасетов из ЛР 2.

5. Оформить отчет по выполненной работе, загрузить исходный код и отчет в соответствующий репозиторий на github.

**Код программы:**

import torch

import torch.nn as nn

import torch.optim as optim

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

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

import pandas as pd

import numpy as np

from ucimlrepo import fetch\_ucirepo

dataset = fetch\_ucirepo(id=186)

X = dataset.data.features

y = dataset.data.targets

print("Размер данных:", X.shape)

print("Первые 5 строк:")

print(X.head())

X = X.fillna(X.mean())

y = y.fillna(y.mean())

le = LabelEncoder()

for col in X.select\_dtypes(include=['object']).columns:

    X.loc[:, col] = le.fit\_transform(X[col].astype(str))

scaler = StandardScaler()

X = scaler.fit\_transform(X)

X = torch.tensor(X, dtype=torch.float32)

y = torch.tensor(y.values, dtype=torch.float32).view(-1, 1)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

class Autoencoder(nn.Module):

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

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

        self.encoder = nn.Sequential(

            nn.Linear(input\_dim, 12),

            nn.ReLU(),

            nn.Linear(12, 6),

            nn.ReLU()

        )

        self.decoder = nn.Sequential(

            nn.Linear(6, 12),

            nn.ReLU(),

            nn.Linear(12, input\_dim)

        )

    def forward(self, x):

        encoded = self.encoder(x)

        decoded = self.decoder(encoded)

        return decoded

input\_dim = X\_train.shape[1]

autoencoder = Autoencoder(input\_dim)

criterion = nn.MSELoss()

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

num\_epochs = 50

for epoch in range(num\_epochs):

    optimizer.zero\_grad()

    outputs = autoencoder(X\_train)

    loss = criterion(outputs, X\_train)

    loss.backward()

    optimizer.step()

    if (epoch + 1) % 10 == 0:

        print(f"Эпоха [{epoch + 1}/{num\_epochs}], Потеря: {loss.item():.4f}")

with torch.no\_grad():

    X\_train\_encoded = autoencoder.encoder(X\_train)

    X\_test\_encoded = autoencoder.encoder(X\_test)

class RegressionModel(nn.Module):

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

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

        self.fc = nn.Sequential(

            nn.Linear(input\_dim, 8),

            nn.ReLU(),

            nn.Linear(8, 1)

        )

    def forward(self, x):

        return self.fc(x)

model = RegressionModel(6)

criterion = nn.MSELoss()

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

num\_epochs = 100

for epoch in range(num\_epochs):

    optimizer.zero\_grad()

    outputs = model(X\_train\_encoded)

    loss = criterion(outputs, y\_train)

    loss.backward()

    optimizer.step()

    if (epoch + 1) % 20 == 0:

        print(f"Эпоха [{epoch + 1}/{num\_epochs}], Потеря: {loss.item():.4f}")

model.eval()

with torch.no\_grad():

    preds = model(X\_test\_encoded).numpy()

    true = y\_test.numpy()

    mse = mean\_squared\_error(true, preds)

    mae = mean\_absolute\_error(true, preds)

    mape = mean\_absolute\_percentage\_error(true, preds)

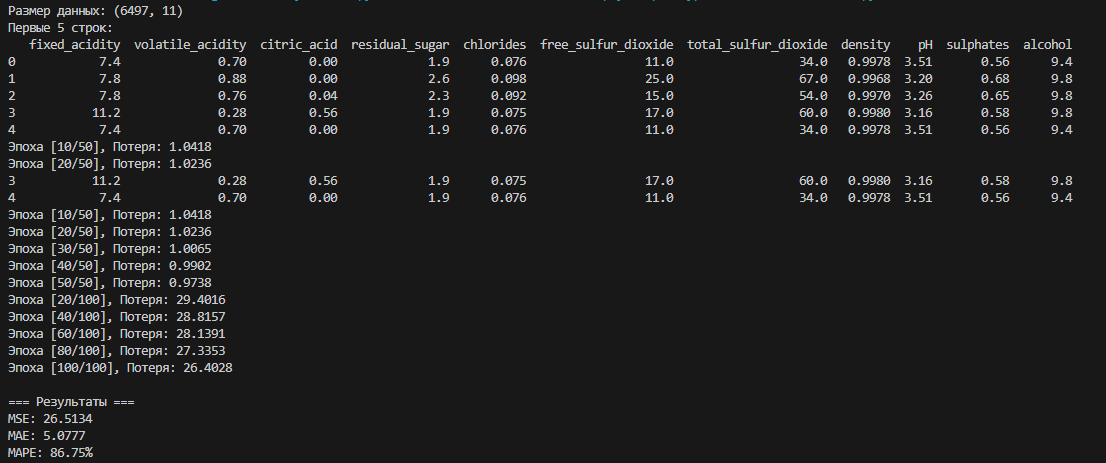
print("\n=== Результаты ===")

print(f"MSE: {mse:.4f}")

print(f"MAE: {mae:.4f}")

print(f"MAPE: {mape \* 100:.2f}%")

**Результат работы программы:**

****

Вывод: научилась осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода.