Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №4
По дисциплине: «ОМО»

Выполнил:
Студент 3-го курса
Группы АС-66
Куган Н. Л.
Проверил:
Крощенко А.А.

Брест 2025

Цель работы: Построить, обучить и оценить многослойный перцептрон (MLP) для решения задачи классификации

Задачи:

1. Загрузить датасет по варианту;
2. Разделить данные на обучающую и тестовую выборки;
3. Обучить на обучающей выборке три модели: k-NN, Decision Tree и SVM;
4. Для модели k-NN исследовать, как меняется качество при разном количестве соседей (k);
5. Оценить точность каждой модели на тестовой выборке;
6. Сравнить результаты, сделать выводы о применимости каждого метода для данного набора данных.

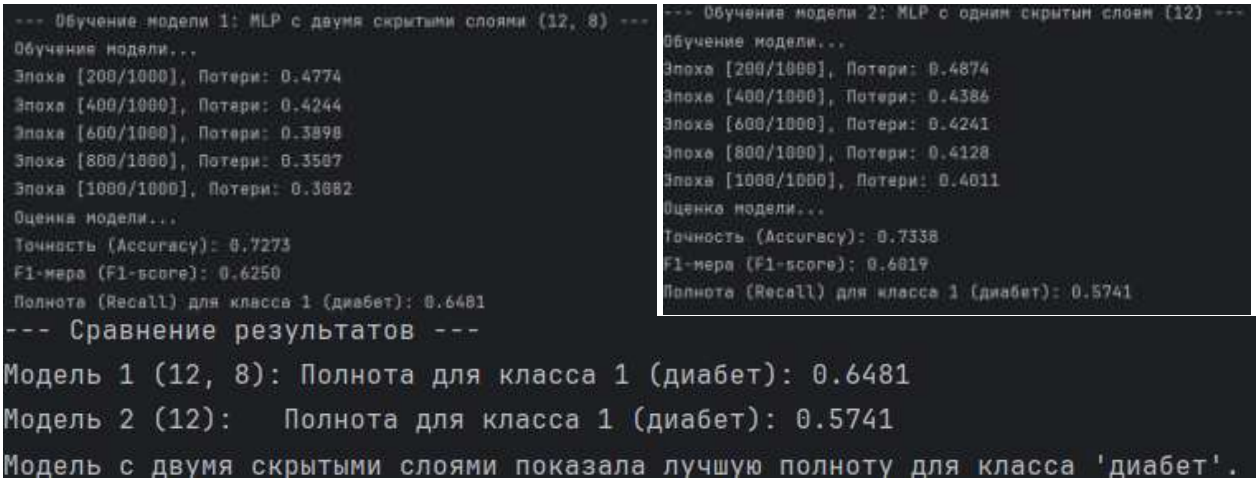
Ход работы

Вариант 6

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, recall_score

import torch
import torch.nn as nn
import torch.optim as optim

# 1. Импорт библиотек и подготовка данных
print("1. Загрузка и подготовка данных...")

column_names = [
    'num_pregnant', 'glucose', 'blood_pressure', 'skin_thickness',
    'insulin', 'bmi', 'diabetes_pedigree', 'age', 'class'
]

df = pd.read_csv('pima-indians-diabetes.csv', names=column_names, header=None)

# Удаляем строки с пропущенными значениями
df = df.dropna()

# Фильтруем только корректные метки
df = df[df['class'].isin([0, 1])]
df['class'] = df['class'].astype(int)

print(f"Осталось строк после очистки: {len(df)}")
print(f"Распределение классов:\n{df['class'].value_counts()}")

X = df.drop('class', axis=1).values
y = df['class'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

X_train_tensor = torch.tensor(X_train_scaled, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test_scaled, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32).unsqueeze(1)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32).unsqueeze(1)
```

```
# 2. Определение архитектуры нейронной сети
class MLP(nn.Module):
    def __init__(self, input_size, hidden1_size, hidden2_size=None, output_size=1):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden1_size)
        self.relu1 = nn.ReLU()
        self.hidden2_exists = hidden2_size is not None
        if self.hidden2_exists:
            self.fc2 = nn.Linear(hidden1_size, hidden2_size)
            self.relu2 = nn.ReLU()
            self.fc3 = nn.Linear(hidden2_size, output_size)
        else:
            self.fc2 = nn.Linear(hidden1_size, output_size)

    def forward(self, x):
        x = self.relu1(self.fc1(x))
        if self.hidden2_exists:
            x = self.relu2(self.fc2(x))
            x = self.fc3(x)
        else:
            x = self.fc2(x)
        return x


# Функция для обучения и оценки модели
def train_and_evaluate_model(model, X_train, y_train, X_test, y_test, epochs=1000,
lr=0.001):
    criterion = nn.BCEWithLogitsLoss()  # Подходит для бинарной классификации с
логитами
    optimizer = optim.Adam(model.parameters(), lr=lr)

    print(f"Обучение модели...")
    for epoch in range(epochs):
        model.train()
        optimizer.zero_grad()
        y_pred = model(X_train)
        loss = criterion(y_pred, y_train)
        loss.backward()
        optimizer.step()

        # Выводим прогресс каждые 200 эпох
        if (epoch + 1) % 200 == 0:
            print(f'Эпоха [{epoch + 1}/{epochs}], Потери: {loss.item():.4f}')

    print("Оценка модели...")
    model.eval()
    with torch.no_grad():
        test_outputs = model(X_test)
        predicted = (torch.sigmoid(test_outputs) > 0.5).float()  # Применяем сигмоиду и
порог

        # Рассчитываем метрики
        accuracy = accuracy_score(y_test.numpy(), predicted.numpy())
        f1 = f1_score(y_test.numpy(), predicted.numpy())
        recall = recall_score(y_test.numpy(), predicted.numpy())

        print(f'Точность (Accuracy): {accuracy:.4f}')
        print(f'F1-мера (F1-score): {f1:.4f}')
        print(f'Полнота (Recall) для класса 1 (диабет): {recall:.4f}')
        print("-" * 40)

        return accuracy, f1, recall, predicted.numpy()


# 3. Инициализация, обучение и оценка первой модели (с двумя скрытыми слоями)
print("\n--- Обучение модели 1: MLP с двумя скрытыми слоями (12, 8) ---")
input_size = X_train.shape[1]  # 8 признаков
```

```
model1 = MLP(input_size=input_size, hidden1_size=12, hidden2_size=8, output_size=1)

# Обучаем и оцениваем модель 1
acc1, f1_1, rec1, pred1 = train_and_evaluate_model(model1, X_train_tensor,
y_train_tensor, X_test_tensor, y_test_tensor)

# 4. Инициализация, обучение и оценка второй модели (с одним скрытым слоем)
print("\n--- Обучение модели 2: MLP с одним скрытым слоем (12) ---")
model2 = MLP(input_size=input_size, hidden1_size=12, hidden2_size=None, output_size=1)

# Обучаем и оцениваем модель 2
acc2, f1_2, rec2, pred2 = train_and_evaluate_model(model2, X_train_tensor,
y_train_tensor, X_test_tensor, y_test_tensor)

# 5. Сравнение моделей
print("\n--- Сравнение результатов ---")
print(f"Модель 1 (12, 8): Полнота для класса 1 (диабет): {rec1:.4f}")
print(f"Модель 2 (12):   Полнота для класса 1 (диабет): {rec2:.4f}")

if rec1 > rec2:
    print("Модель с двумя скрытыми слоями показала лучшую полноту для класса
'диабет'.")
elif rec2 > rec1:
    print("Модель с одним скрытым слоем показала лучшую полноту для класса 'диабет'.")
else:
    print("Обе модели показали одинаковую полноту для класса 'диабет'.")
```



Вывод: Я построил, обучил и оценил многослойный перцептрон (MLP) для решения задачи классификации